Refine Search

Search Results -

Terms	Documents
707/1	9059

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: Refine Search

Recall Text Clear Interrupt

Search History

DATE: Thursday, February 01, 2007 Purge Queries Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L33	707/1	9059	L33
L32	707/10	14089	L32
L31	707/100	9542	L31
L30	707/101	5782	L30
L29	707/104.1	7678	L29
L28	707/200	5605	L28
L27	707/201	3679	L27
L26	707/206	1467	L26
L25	707.clas.	40851	L25
L24	705.clas.	48000	L24
L23	705/1	6742	L23
L22	705/5	1087	L22
L21	705/7	2891	L21

L20	705/14	5127	L20
L19	705/16	1120	L19
L18	705/26	7072	L18
L17	705/28	2157	L17
L16	705/30	1209	L16
L15	705/35	2870	L15
L14	705/39	2140	L14
L13	705/44	1285	L13
L12	706/52	584	L12
L11	706.clas.	7880	L11
L10	717.clas.	13162	L10
L9	717/102	246	L9
L8	717/104	763	L8
L7	717/105	469	L7
L6	L5 and (stag$ near5 tables or temporary near5 tables)	13	L6
L5	L4 and business with database	63	L5
L4	L3 and (populat$ and datamart or populat$ amd data near2 mart)	82	L4
L3	L2 and metadata	119	L3
L2	L1 and (business and database or business and data near2 base)	221	L2
L1	"star schema" and (datawarehouse or data with warehouse or data with mart or datamart)	266	L1

END OF SEARCH HISTORY

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

☐ Generate Collection Print

L6: Entry 12 of 13 File: USPT Feb 13, 2001

US-PAT-NO: 6189004
DOCUMENT-IDENTIFIER: US 6189004 B1

TITLE: Method and apparatus for creating a datamart and for creating a query structure for the datamart

DATE-ISSUED: February 13, 2001

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Rassen; Jeremy A.	Sunnyvale	CA		
Litvak; Emile	Mountain View	CA		
shelat; abhi a.	Mountain View	CA		
McCaskey; John P.	Mountain View	CA		
Rauer; Allon	Mountain View	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
E. Piphany, Inc.	San Mateo	CA			02

APPL-NO: 09/073753 [PALM]
DATE FILED: May 6, 1998

PARENT-CASE:
CROSS REFERENCES TO RELATED APPLICATIONS This application relates to the following group of applications. Each application in the group relates to, and incorporates by reference, each other application in the group. The invention of each application is assigned to the assignee of this invention. The group of applications includes the following. U.S. patent application Ser. No. 09/073,748, entitled "Method and Apparatus for Creating a Well-Formed Database System Using a Computer," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh, and Eliot Leonard Wegbreit. U.S. patent application Ser. No. 09/073,752, entitled "Method and Apparatus for Creating and Populating a Datamart," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Lynn Randolph Slater, Jr. U.S. patent application Ser. No. 09/073,733, entitled "Method and Apparatus for Creating Aggregates for Use in a Datamart," filed May 6, 1998, and having inventors Allon Rauer, Gregory Vincent Walsh, John P. McCaskey, Craig David Weissman and Jeremy A. Rassen. U.S. patent application Ser. No. 09/073,753, entitled "Method and Apparatus for Creating a Datamart and for Creating a Query Structure for the Datamart," filed May 6, 1998, and having inventors Jeremy A. Rassen, Emile Litvak, abhi a. shelat, John P. McCaskey and Allon Rauer.

INT-CL-ISSUED: [07] G06F 17/30

INT-CL-CURRENT:

TYPE IPC DATE

CIPP G06 F 17/30 20060101

US-CL-ISSUED: 707/3; 707/4, 707/102
US-CL-CURRENT: 707/3; 707/102, 707/4

FIELD-OF-CLASSIFICATION-SEARCH: 707/1-10, 707/100-104, 707/200-206
See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL | Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5386556	January 1995	Hedin et al.	707/4
☐	5550971	August 1996	Brunner et al.	707/3
☐	5659724	August 1997	Borgida et al.	707/3
☐	5675785	October 1997	Hall et al.	707/102
☐	5806060	September 1998	Borgida et al.	707/3
☐	5995958	November 1999	Xu	707/3

OTHER PUBLICATIONS

Kimball, R., "The Data Warehouse Toolkit", (1996) John-Wiley & Sons, Inc., 388 pages (includes CD ROM).
Chawathe, S. et al., "Change Detection in Hierarchically Structured Information", SIGMOD Record, vol. 25, No. 2, Jun. 1996, pp. 493-504.
Chawathe, S. et al., "Meaningful Change Detection in Structured Data", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 26-37.
Labio, W. et al., "Efficient Snapshot Differential Algorithms for Data Warehousing", Department of Computer Science, Stanford University, (1996), pp. 1-13.
Wiener, J. et al., "A System Prototype for Warehouse View Maintenance", The Workshop on Materialized Views, pp. 26-33, Montreal, Canada, Jun. 1996.
Kawaguchi, A. et al., "Concurrency Control Theory for Deferred Materialized Views", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 306-320.
Zhuge, Y. et al., "Consistency Algorithms for Multi-Source Warehouse View Maintenance", Distributed and Parallel Databases, vol. 6, pp. 7-40 (1998), Kluwer Academic Publishers.
Zhuge, Y. et al., "View Maintenance in a Warehousing Environment", SIGMOD Record, vol. 24, No. 2, Jun. 1995, pp. 316-327.
Widom, J., "Research Problems in Data Warehousing", Proc. of 4th Int'l Conference on Information and Knowledge Management (CIKM), Nov. 1995, 6 pages.
Yang, J. et al., "Maintaining Temporal Views Over Non-Historical Information Sources For Data Warehousing", Advances in Database Technology--EDBT '98, Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, Mar. 1998, pp. 389-403.
Quass, D., "Maintenance Expressions for Views with Aggregation", Proceedings of the 21st International Conference on Very Large Data Bases, IEEE, Zurich, Switzerland, (Sep. 1995), 9 pages.

Mumick, I. et al., "Maintenance of Data Cubes and Summary Tables in a Warehouse", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 100-111..
Huyn, N., "Multiple-View Self-Maintenance in Data Warehousing Environments", Proceedings of the 23rd International Conference on Very Large Data Bases, IEEE, (1997), pp. 26-35.
Quass, D. et al., "Making Views Self-Maintainable for Data Warehousing", Proceedings of the Fourth International Conference on Parallel and Distributed Information Systems, IEEE, Dec. 1996, pp. 158-169.
Quass, D. et al., "On-Line Warehouse View Maintenance", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 393-404.
Gupta, H., "Selection of Views to Materialize in a Data Warehouse", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 98-112.
Harinarayan, V. et al., "Implementing Data Cubes Efficiently", SIGMOD Record, vol. 25, No. 2, Jun. 1996, pp. 205-216.
Gupta, H. et al., "Index Selection for OLAP", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 208-219.
Labio, W. et al., "Physical Database Design for Data Warehouses", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 277-288.
Gupta, A. et al., "Aggregate-Query Processing in Data Warehousing Environments", Proceedings of the 21st VLDB Conference, Zurich, Switzerland, Sep. 1995, pp. 358-369.
O'Neill, P. et al., "Improved Query Performance with Variant Indexes", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 38-49.
McAlpine, G. et al., "Integrated Information Retrieval in a Knowledge Worker Support System", Proc. of the Intl. Conf. on Research and Development In Information Retrieval (SIGIR), Cambridge, MA, Jun. 25-28, 1989, Conf. 12, pp. 48-57.
Tsuda, K. et al., "IconicBrowser: An Iconic Retrieval System for Object-Oriented Databases", Proc. of the IEEE Workshop on Visual Languages, Oct. 4, 1989, pp. 130-137.
"Multiple Selection List Presentation Aids Complex Search", IBM Technical Disclosure Bulletin, vol. 36, No. 10, Oct. 1993, pp. 317-318.

ART-UNIT: 271

PRIMARY-EXAMINER: Ho; Ruay Lian

ATTY-AGENT-FIRM: Wilson Sonsoni Goodrich & Rosati

ABSTRACT:

A method for automatically defining a query interface for a datamart is described. The datamart includes fact and dimension tables. The method comprises accessing a schema description and a query interface description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The query interface description specifies the fields, related to the schema description, that can be used in a query and the way in which results are to be presented to the user. The fields correspond to columns and rows in the fact tables. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create the query interface is created. Some commands of the first set of commands are executed causing the creation and population of the tables. Some commands of the second set of commands are executed causing the creation of a user interface. A query is generated using the user interface. The query is sent to the system for processing. The results of the query are presented to the user according the second set of

commands.

9 Claims, 43 Drawing figures

Previous Doc Next Doc Go to Doc#

L6: Entry 12 of 13 File: USPT Feb 13, 2001

DOCUMENT-IDENTIFIER: US 6189004 B1
TITLE: Method and apparatus for creating a datamart and for creating a query structure for the datamart

Abstract Text (1):
A method for automatically defining a query interface for a datamart is described. The datamart includes fact and dimension tables. The method comprises accessing a schema description and a query interface description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The query interface description specifies the fields, related to the schema description, that can be used in a query and the way in which results are to be presented to the user. The fields correspond to columns and rows in the fact tables. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create the query interface is created. Some commands of the first set of commands are executed causing the creation and population of the tables. Some commands of the second set of commands are executed causing the creation of a user interface. A query is generated using the user interface. The query is sent to the system for processing. The results of the query are presented to the user according the second set of commands.

Parent Case Text (3):
U.S. patent application Ser. No. 09/073,748, entitled "Method and Apparatus for Creating a Well-Formed Database System Using a Computer," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh, and Eliot Leonard Wegbreit.

Parent Case Text (4):
U.S. patent application Ser. No. 09/073,752, entitled "Method and Apparatus for Creating and Populating a Datamart," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Lynn Randolph Slater, Jr.

Parent Case Text (5):
U.S. patent application Ser. No. 09/073,733, entitled "Method and Apparatus for Creating Aggregates for Use in a Datamart," filed May 6, 1998, and having inventors Allon Rauer, Gregory Vincent Walsh, John P. McCaskey, Craig David Weissman and Jeremy A. Rassen.

Parent Case Text (6):
U.S. patent application Ser. No. 09/073,753, entitled "Method and Apparatus for Creating a Datamart and for Creating a Query Structure for the Datamart," filed May 6, 1998, and having inventors Jeremy A. Rassen, Emile Litvak, abhi a. shelat, John P. McCaskey and Allon Rauer.

Brief Summary Text (4):
This invention relates to the field of databases. In particular, the invention relates to creating databases, and loading and accessing data in the databases.

Brief Summary Text (6):

Many different types of databases have been developed. On line transaction processing 3.- Ups (OLTP) databases are examples of typical databases used today. OLTP databases are concerned with the transaction oriented processing of data. On line transaction processing is the process by which data is entered and retrieved from these databases. In these transaction-oriented databases, every transaction is guaranteed. Thus, at a very low level, the OLTP databases are very good at determining whether any specific transaction has occurred.

Brief Summary Text (7):
Another type of database is a data warehouse or datamart. A datamart transforms the raw data from the OLTP databases. The transformation supports queries at a much higher level than the OLTP atomic transaction queries. A data warehouse or a datamart typically provides not only the structure for storing the data extracted from the OLTP databases, but also query analysis and publication tools.

Brief Summary Text (8):
The advantage of datamarts is that users can quickly access data that is important to their business decision making. To meet this goal, datamarts should have the following characteristics. First, datamarts should be consistent in that they give the same results for the same search. The datamart should also be consistent in the use of terms to describe fields in the datamart. For example, "sales" has a specific definition, that when fetched from a database, provides a consistent answer. Datamarts should also be able to separate and combine every possible measure in the business. Many of these issues are discussed in the following book, Ralph Kimball, The Data Warehouse Toolkit, John Whiley and Sons, Inc., New York, N.Y. (1996).

Brief Summary Text (9):
Multi-dimensional datamarts are one kind of datamart. Multi-dimensional datamarts rely on a dimension modeling technique to define the schema for the datamart. Dimension modeling involves visualizing the data in the datamart as a multi-dimension data space (e.g., image the data as a cube). Each dimension of that space corresponds to a different way of looking at the data. Each point in the space, defined by the dimensions, contains measurements for a particular combination of dimensions. For example, a three dimensional cube might have product, customer, and territory dimensions. Any point in that cube, defined by those three dimensions, will represent data that relates those three dimensions.

Brief Summary Text (10):
The data in the datamart is organized according to a schema. In a dimensional datamart, the data is typically organized as a star schema. At the center of a standard star schema is a fact table that contains measure data. Radiating outward from the fact table, like the points of a star, are multiple dimension tables. Dimension tables contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are joined. The advantage of such a schema is that it supports a top down business approach to the definition of the schema.

Brief Summary Text (11):
Present datamarts have a number of drawbacks that are now discussed. First, datamarts are typically difficult to build and maintain. This is because of the requirements that they be consistent and flexible. A related drawback of present day datamarts is that they do not allow the consultants of the datamart to make changes to the schema simply and easily. Because datamarts support very high level queries about the business processes in the business, they require a great deal of consistency in the use of data from the OLTP systems. Additionally, the datamarts need to be very flexible to address changes in the types of high level queries supported. Changing typical datamarts require the changing of hundreds, or

potentially thousands, of lines of SQL code. For example, if a fact column is added to a fact table, the change propagates throughout the datamart. These changes are typically implemented by hand, a very time consuming and error prone process. As a result of the hand coding involved, it is quite possible to construct the database in an arbitrary fashion that does not conform to good rules for constructing datamarts. Thus, well-formed datamarts may not result.

Brief Summary Text (14):
One embodiment of the invention includes a method for automatically defining a query interface for a datamart. The datamart includes fact and dimension tables. The method comprises accessing a schema description and a query interface description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The query interface description specifies the fields, related to the schema description, that can be used in a query and the way in which results are to be presented to the user. The fields correspond to columns and rows in the fact tables. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create the query interface is created. Some commands of the first set of commands are executed causing the creation and population of the tables. Some commands of the second set of commands are executed causing the creation of a user interface. A query is generated using the user interface. The query is sent to the system for processing. The results of the query are presented to the user according the second set of commands.

Drawing Description Text (3):
FIG. 1 illustrates a datamart system representing one embodiment of the invention.

Drawing Description Text (4):
FIG. 2 illustrates an embodiment of a method of defining the datamart, loading the datamart, and then querying the data.

Drawing Description Text (5):
FIG. 3 illustrates a schema used in the system of FIG. 1 to define schemas for the datamart.

Drawing Description Text (9):
FIG. 7 through FIG. 29 describe a user interface that can be used to define a schema, build a datamart, load the datamart, and query the datamart.

Detailed Description Text (13):
DATAMART SYSTEM

Detailed Description Text (16):
Metadata Overview

Detailed Description Text (19):
EXAMPLE METHOD OF DEFINING AND USING THE DATAMART

Detailed Description Text (20):
TOP LEVEL METADATA SCHEMA

Detailed Description Text (21):
Top Level Metadata List

Detailed Description Text (22):
Top Level Metadata Descriptions

Detailed Description Text (30):
Top Level Metadata Use

Detailed Description Text (31):
EXTRACTION METADATA

Detailed Description Text (32):
Extraction Metadata List

Detailed Description Text (33):
Extraction Metadata Descriptions

Detailed Description Text (41):
Extraction Metadata Use

Detailed Description Text (45):
RUNTIME METADATA

Detailed Description Text (46):
Runtime Metadata List

Detailed Description Text (47):
Runtime Metadata Descriptions

Detailed Description Text (49):
QUERY MECHANISM METADATA

Detailed Description Text (51):
Query Mechanism Schema Metadata Descriptions

Detailed Description Text (52):
Ticksheets Metadata

Detailed Description Text (53):
Measurement Metadata

Detailed Description Text (54):
Filtering Metadata

Detailed Description Text (55):
Display Options Metadata

Detailed Description Text (56):
USER INTERFACE EXAMPLE OF DEFINING METADATA

Detailed Description Text (66):
The following describes a system according to various embodiments of the invention. Generally, the system allows a consultant to define a well-formed datamart. The system includes tables and columns that conform to the definition of the datamart. The system also includes additional columns for foreign key tracking, source system key mapping, time and date tracking. The system has automatic indexing. The system enforces typing information about the data stored in the datamart. These additional features cause the datamart to operate in a consistent manner. One benefit of such consistent operation is that results are consistent in meaning from query to query.

Detailed Description Text (67):
Focusing on the datamart creation, the system allows a consultant to build a datamart from a schema definition and a definition of the sources of the data. From the schema definition, the system automatically builds the tables needed in the datamart. Also, from the schema definition, and the sources definition, the system can automatically extract the data from those sources. Depending on the semantic

meaning of the data, as defined by the schema definition, the system automatically converts the data from the sources into forms that are readily usable in the datamart. Once the datamart has been created, and the data has been loaded, users can then perform queries on the data.

Detailed Description Text (68):
As part of the datamart creation, the system allows the consultant to define aggregates for the datamart. The aggregates correspond to pre-computed query results for different types of queries. For example, an aggregate can be created for a query that asks for all sales, by region, by quarter. The corresponding aggregate table would include a set of rows that have the results for this query (e.g., each row includes the quarterly sales for each region). The aggregates are specified using the schema definition. This makes defining and changing aggregates relatively simple.

Detailed Description Text (69):
To allow a user to query the datamart, the system includes an interface for defining what fields can be used by the user to query the datamart. Additionally, by allowing the consultant to define measure and related information, the system allows the consultant to specify how the results are to appear to the users.

Detailed Description Text (70):
The following description first presents a system level view of primarily one embodiment. Then, an example use of the system is presented. Next, the metadata used in the system is described. This metadata description is broken into four parts: a top level description of the metadata used in defining schemas, a description of the metadata used during the extraction, a description of the metadata used while the datamart is running, and a description of the query interface metadata. Next, an example set of user interface screen shots illustrates how consultants can quickly and efficiently define schemas, aggregates, and query interfaces, and how users can query the datamart. Next, additional alternative embodiments are described.

Detailed Description Text (72):
Datamart or Data Warehouse--is a database.

Detailed Description Text (73):
Schema--is a description of the organization of data in a database. Often, the schema is defined using a data definition language provided by a database management system. More abstractly, the schema can be the logical definition of a data model for use in a database.

Detailed Description Text (74):
Metadata--is data that defines other data. This is not the actual data in the datamart, but is the data that defines the data in the datamart.

Detailed Description Text (75):
Constellation--a grouping of dimension definitions, fact definitions, like-structured facts (all facts in a constellation have the same dimensional foreign keys), or stars, and other metadata definitions. Often the grouping relates to a business process (e.g., sales).

Detailed Description Text (76):
Fact Table--the central table of a star schema. It stores the numeric measurements of the business that is supplying the information to the datamart.

Detailed Description Text (78):
Dimension--the tables that link to the fact table in a star schema. The tables store the descriptions of the dimensions of the business. Examples of dimensions are product and territory.

Detailed Description Text (80):
User--any end user who would normally wish to query a datamart, but would not usually be concerned with the implementation or maintenance of the datamart.

Detailed Description Text (81):
Consultant--is a person responsible for the creation and maintenance of a datamart.

Detailed Description Text (82):
Source System--is any computer system that holds the raw data used by the system. Examples of such source systems are OLTP database systems.

Detailed Description Text (83):
Data Store--any data storage (physical or logical) from which data is received or to which data is stored. Examples of a data store are files, a database, etc.

Detailed Description Text (86):
Datamart System

Detailed Description Text (87):
FIG. 1 illustrates a datamart system representing one embodiment of the invention. The system supports the creation of a well-formed datamart. This system allows consultants to use metadata to define schemas for a datamart. From the definition of the schema, the system can automatically generate the tables in the datamart. Further, the system can automatically extract the data from the source systems, perform conversions on that data and populate the datamart. The system supports the automatic creation and processing of aggregates from aggregate definitions. The system also supports the creation of the query mechanisms from query definitions.

Detailed Description Text (90):
FIG. 1 includes the following elements: source systems 110, a system 100, a web server 186, a consultant computer 190, and a user computer 180. The system 100 includes the metadata 160, an enterprise manager 102, an extraction program 120, staging tables 130, a semantic template conversion program 140, a datamart 150, an aggregate builder 170, and a query and reporting program 104. The metadata 160 includes the following data: schema definitions 161, connectors 162 (connectors are also referred to as extractors), semantic definitions 163, source system information 164, aggregate information 167, measurement information 168, and query/reporting information 169. The user computer 180 is shown running a browser 182. The browser 182 includes a query/results interface 184. The consultant computer 190 shows the enterprise manager interface 192 which shows the metadata organization of the system 100.

Detailed Description Text (92):
The following describes the metadata 160, then the other elements of the system 100, and finally, the elements that are external to the system 100. These elements are all described in greater detail below.

Detailed Description Text (93):
Metadata Overview

Detailed Description Text (94):
The metadata 160 includes many different types of data and information. This information can be broken down into information related to (1) the definition of the schema for the datamart 150, (2) the data needed during the extraction from the source systems 110 and loading of the datamart 150, and (3) the information used in the querying of the datamart 150 and supplying the result sets. The relationships between the elements of the metadata 160 are described in greater detail below. However, the following provides brief descriptions of these elements.

Detailed Description Text (95):
The schema definitions 161 hold the definition of the schema for the datamart 150. Typically, a consultant, using the consultant computer 190, can interface with the enterprise manager 102 to define the schema definition 161 for the datamart 150. In particular, the consultant can use the enterprise manager interface 192 to define a star schema for the datamart 150. This star schema is organized around the business processes of the business for which the datamart is being created. What is important is that the consultant can easily define a schema for the datamart 150 and that definition is kept in the schema definitions 161. From the schema definitions 161, not only can the tables in the datamart 150 be generated, but also the automatic extraction and conversion of the data from the source systems 110 can be performed, aggregates are set up, and a query mechanism is generated.

Detailed Description Text (96):
The connectors 162, the semantic definitions 163, and the source system information 164, are all related to the extraction of the data from the source systems 110. The connectors 162 define the access routines for extracting the source systems data 110. The semantic definitions 163 define how that extracted data should be converted when it is loaded into the datamart 150. The semantic definitions 163 provide important advantages to the system 100. In particular, the semantic definitions 163 allow for a simplified definition of the datamart 150, consistent meaning of the data in the datamart 150, and allow for complex changes to the schema to be easily propagated to the datamart 150. The source system information 164 defines how to extract the data from the systems 110.

Detailed Description Text (97):
The aggregate information 167 defines how data in the datamart 150 is treated once it is extracted. The aggregate information 167 allows for the creation of aggregates. Aggregates are aggregations of various fields of data in the datamart 150. Aggregates support more complex and powerful queries to be executed on the datamart 150. The aggregates also improve the performance of the system during the querying process and allow for time navigation of the data in the datamart 150. Time navigation is the process of creating backlog result sets by hopping through date aggregates from the beginning of time in the datamart 150 to the present.

Detailed Description Text (98):
The measurement information 168 and the query/reporting information 169 support the querying of the datamart 150. A measure is a piece of numeric data in the datamart 150 that is useful to a user. That is, individual fact columns from source systems can be very implementation specific. These columns may not correspond to what users would prefer to see. For example, a user may want to see a net price added with a total cost. However, the fact table may only include the net price or the total cost. The measure information 168 allows the consultant to define the abstract notion of the calculation associated with the net price added to the total cost.

Detailed Description Text (99):
In some embodiments of the invention, the metadata 160 also includes security information. The security information defines the level of access for various users to the various tables and fields in the datamart 150. This security information automatically restricts access to that data.

Detailed Description Text (101):
The system 100 can be implemented on a network of computers running Windows NT and son UNIX. The datamart 150 can be implemented on top of an Oracle (SQL Server, or ODBC) database. However, this physical structure of the system 100 can be implemented in any number of ways, and the invention does not require this specific hardware configuration.

Detailed Description Text (102):

The enterprise manager 102 is a program that is responsible for supporting the definition of the schema, and the creation of the tables in the datamart 150 from the schema definitions 161. The enterprise manager 102 also controls the extraction program 120. (In some embodiments, the extraction program 120 and the semantic template conversion program 140 are included in the enterprise manager 102). During the execution of the extraction program 120, the extraction program 120, the staging tables 130, the semantic template conversion 140, and the datamart 150 are all used. The extraction program 120 uses the connectors 162 and the source system information 164 to extract the information from the source systems 110. The extracted data is loaded into the staging tables 130.

Detailed Description Text (103):
The staging tables 130 are temporary tables used to hold the source system data before performing any semantic conversions on that data. The staging tables 130 also allow for the conversion of the source system data prior to moving the data into the datamart 150. Once the staging tables 130 have been loaded, the semantic definitions 163 can be accessed from the enterprise manager 102 to convert the information in the staging tables 130 to predefined data semantics. These predefined data semantics allow for powerful queries, consistency in the definition of the meaning of the data in the datamart 150, and allow for changes to be made to the schema. Generally, the semantic template conversion 140 takes data stored in the staging tables 130, performs a conversion of that data according to a corresponding semantic definition (defined in the schema definitions 161), and populates the datamart 150 with the converted data.

Detailed Description Text (104):
Importantly, the predefined data semantics substantially simplify the creation and population of the datamart 150. In previous systems, the consultant would have to implement all of the data manipulation and population programs by hand. By selecting a particular semantic definition for a particular fact, or dimension, in the schema, the consultant has automatically defined the access and manipulation for populating programs for that table. Allowing the consultant to select a predefined data semantic not only reduces the tedious coding previously required of the consultant, but also allows for the automatic insertion of foreign keys, transaction types, date, and other information into the schema, and therefore the datamart 150. This additional information causes the datamart 150 to be well-formed.

Detailed Description Text (105):
The aggregate builder 170, as mentioned above, aggregates data in the datamart 150 according to the aggregate information 167 and the schema definitions 161. The results of the aggregate builder 170 allow for more powerful and faster queries to be performed on the datamart 150.

Detailed Description Text (106):
The query/reporting program 104 supports the querying of the datamart 150 and presents results of those queries. The query and reporting process 104 uses the measurement information 168 and the query and reporting information 169, in addition to the schema definitions 161, to query the datamart 150 and provide that information to the web server 186. The query/reporting information 169 includes filters and form definitions. The filters allow the user to filter different fields out of the datamart 150. The forms allow the users to indicate which fields a user is particularly interested in.

Detailed Description Text (107):
The metadata 160, although including many different types of definitional data, importantly includes the schema definition 161 and the semantic definitions 163. The enterprise manager 102 can use the schema definitions 161 to build the tables in the datamart 150. Through the combination of these two pieces of metadata 160, the enterprise manager 102 can take data from a source system 110, perform semantic

conversions on that data and populate the datamart 150. Thus, in some embodiments of the invention, the system includes only the schema definitions 161 and the semantic definitions 163.

Detailed Description Text (109):
The source systems 110, as defined above, represent large databases from which data for the datamart 150 is pulled. Examples of such systems include large on line transaction processing (OLTP) systems. Typically these source systems 110 are relational databases having multiple tables and relations between those tables. The source systems 110 do not generally support powerful queries that provide high level information about the business in which the source systems 110 are used. Thus, the system 100 is used to extract the data from the source systems 110 and to provide an improved schema for querying that data. In some embodiments, the source systems 110 include non-relational databases such as object databases. In other embodiments, the source systems 110 can include flat file systems or combined relational and object databases. What is important is the source systems 110 can provide data to the system 100 through the connectors 162 and the source system information 164.

Detailed Description Text (111):
The user can access the web server 186 through the user computer 180. In this example, the user computer 180 can access the web server 186 through an HTTP connection. The user computer 180 sends a file request to the web server 186. This file request represents a request for a query of the datamart 150. The web server 186 runs a Java program upon receiving the request. The query and reporting program 104 converts the information from the Java program into a query that the datamart 150 will understand. In one embodiment, the query/reporting program 104 converts the query into a set of SQL statements. The SQL statements are run against the datamart 150. The results of the statements are processed and provided back to the user computer 180.

Detailed Description Text (112):
Example Method of Defining and Using the Datamart

Detailed Description Text (113):
FIG. 2 illustrates an embodiment of a method of defining the datamart 150, loading the datamart 150, and then accessing the data in the datamart 150. This example can be broken into four subparts: a build datamart process 202, an extraction and loading process 204, a build aggregates process 205, and a query and reporting process 206. This example can be implemented using the system 100.

Detailed Description Text (114):
At block 210, a consultant uses the enterprise manager 102 to define the schema. The schema is defined using the metadata 160. This process is illustrated in greater detail in FIG. 7 through FIG. 35. Generally, defining the schema involves determining the business processes of the organization for which the system 100 is being implemented. The consultant then defines the star schema for those business processes. The star schema has a fact table and a number of dimensions. The consultant also defines from where the data in the schema is to be derived. That is, the consultant defines from which fields and tables the information is to be extracted from the source systems 110. The consultant also defines how that data is to be put into the datamart 150. That is, the consultant associates each piece of data with a semantic meaning. This semantic meaning defines how the data from the source system is to be manipulated and how it is to populate the datamart 150. At this point, the consultant can also define the aggregates that can be used in the datamart 150.

Detailed Description Text (115):
Once the datamart 150 has been defined, it can then be automatically built. At block 220, the enterprise manager 102 generates table creation SQL statements

according to the definition of the metadata. In one embodiment of the invention, block 220 is accomplished by performing queries on the schema definitions 161 to generate the fact table creation statements, the fact staging table creation statements, the dimension table creation statements, the dimension staging table creation statements, and the dimension mapping table creation statements. These tables are described in greater detail below. From the results of these queries, SQL CREATE TABLE statements are created. Importantly, the schema definitions 161 provide the information the enterprise manager 102 needs to build the datamart 150.

Detailed Description Text (116):
Note that this process can also be used to modify the schema of an existing datamart 150. Therefore, at block 220, the SQL tables being created will cause the existing datamart 150 to be modified without losing the data in the datamart 150.

Detailed Description Text (117):
At block 230, the enterprise manager 102 issues the table generation statements to the database upon which the datamart 150 is being created. That database creates the tables, which correspond to the datamart 150. After block 230, the build the datamart process 202 is complete.

Detailed Description Text (118):
Now the extraction process 204 can be performed. The extraction process 204 is run on a periodic basis to load data from the source systems 110 into the datamart 150. This process can be run multiple times for the datamart 150.

Detailed Description Text (119):
At block 260, the connectors 162 are used by the enterprise manager 102, and in particular, they are used by the extraction program 120 to extract the data from the source systems 110. The connectors 162 can include SQL statement templates (not to be confused with semantic templates, as described below) for extracting data from the source systems 110. The extraction program 120 uses these templates, in addition to the source system information 164, to generate SQL statements. These SQL statements are issued to the source system 110 and the results are loaded into the staging tables 130. (The staging tables 130 had been created as a result of block 230.) Once the staging tables have been loaded, the data can then be moved into the datamart 150.

Detailed Description Text (120):
At block 270, the staging table data is moved into the datamart 150 using the semantic definitions 163. The semantic definitions 163 are templates for converting the staging tables 130 data according to predefined data semantics. These predefined data semantics, as described below, provide semantic meaning to the data being loaded from the staging tables 130. Note that the data from the staging tables 130, as processed by the semantic template conversion 140, is placed in the tables in the datamart 150.

Detailed Description Text (121):
Thus, the schema definition and the semantic definitions 163 are used to generate and populate the datamart 150 such that the datamart 150 is well-formed. Examples of the well-formedness of the datamart 150 are as follows. (1) Two columns related by a relational join will be from the same domain. (2) If table A has a many-to-one relationship to table B, then table A has a foreign key that corresponds to table B. (3) A many-to-many relationship, between two tables A and B, is always expressed by an associative table that is created in a uniform way. For each unique many-to-many relationship, a unique value is created in the associative table and reused whenever that many-to-many relationship occurs. Denormalization is always done correctly. (4) Pulling information from one table to be put into another table, for access efficiency, is done correctly. Previous systems cannot guarantee such a well-formed database system because hand coding of the creation and population

operations is required. This hand coding can easily introduce errors into datamart creation and population processes. Once the extraction process 204 has completed, the aggregates can be built in the build aggregates process 205. The aggregates are tables of pre-calculated combinations of dimensions and facts. Importantly, they greatly increase the speed of queries. Generally, the aggregate definitions, stored in the aggregate information 167, are accessed and built using the aggregate definitions (which interface with the schema definitions). At block 275, the aggregate builder 170 accesses the metadata 160 to build the aggregates. Often, the aggregate building is done at night.

Detailed Description Text (122):
After aggregates are built, the querying and reporting process 206 can be performed. The querying and reporting process 206 can be performed anytime after the creation of the datamart 150. Importantly, when an aggregate is created, the appropriate operation for that aggregate is used. For example, revenue elements are added to produce an aggregate, while daily account balances are averaged to produce an aggregate.

Detailed Description Text (123):
At block 277, the consultant defines the query mechanism schema for the system 100. In particular, the consultant defines the query/reporting information 169 and the measurement information 168. These two pieces of metadata 160 allow the system 100 to report meaningfully consistent information to users. Also, the consultant is not burdened with having to hand create the possible reports.

Detailed Description Text (124):
At block 280, a query is generated. In one embodiment of the invention the query is generated at the query/reporting program 104. In other embodiments, the query can be generated at the user computer 180 through the HTTP, web server 186, Java coupling to the query/reporting program 104. What is important here is that some query is generated that can be used to access the datamart 150. Importantly because the schema definitions 161 are available to the query and reporting program 104, the user can be presented with forms from which a query can be easily and automatically generated.

Detailed Description Text (125):
At block 290, the answer set (the results) is created by the datamart 150. This answer set is then propagated back through the query/reporting program 104, and ultimately to the user computer 180. The results are formatted according to the query/reporting information 169.

Detailed Description Text (126):
Top Level Metadata Schema

Detailed Description Text (127):
As noted in the background, multi-dimensional datamarts use star schemas. The system 100 uses star schemas in a larger organization that allows for the sharing of dimension tables by sets of similar facts. This larger organization is called a constellation. FIG. 3 illustrates a schema for the schema definitions tables that support constellations. (The schema of FIG. 3 is labeled the schema for schema definitions 300.) That is, FIG. 3 illustrates a schema used in the system 100 to define schemas for the datamart. FIG. 3 also illustrates some of the aggregate information 167 schema.

Detailed Description Text (129):
It is important to remember that FIG. 3 through FIG. 5 illustrate the schema of the system used to generate and run the datamart 150. Rows in these tables define the schema for use in the datamart 150. From these rows, create table, table query, etc., commands are created. These commands are used to create the tables in the datamart 150 and to access that datamart.

Detailed Description Text (130):
Also, as mentioned previously, the datamart 150 is well-formed because, among other reasons, the system 100 automatically includes additional columns in the table created in the datamart 150. For example, source system key, foreign key, and time and date columns are automatically added (where appropriate). The rest of the elements of the system can then rely on the existence of these columns. This prevents, for example, the creation of an inconsistent schema where only some of the tables include date and time information.

Detailed Description Text (132):
Top Level Metadata List

Detailed Description Text (134):
Top Level Metadata Descriptions

Detailed Description Text (135):
It is important to remember that the tables in FIG. 3 are only used to define the schema in the datamart 150. Thus, a fact table 304 in FIG. 3 is not the actual fact table in the datamart 150, but the definition of that fact table. Each row in a table corresponds to an instance of that table.

Detailed Description Text (136):
The constellation 302 defines the organization of the schema in the datamart 150. It is the top level table in the schema definition.

Detailed Description Text (138):
The fact table 304 defines the metadata 160 table describing all of the fact tables within a given constellation 302. The attributes of the fact table 304 include a build aggregates flag, a cleanse flag, a constellation key, a description, a fact table key, a fact table name, and a truncate stage flag. Each attribute corresponds to a column in the fact table 304. The build aggregate flag indicates whether or not to build aggregates for a particular fact on the next execution of the aggregate builder 170. The cleanse flag is a flag that is used in many of the tables to obliterate the actual measures within a table in the datamart 150 (particularly useful in demonstrations of the system 100 where sensitive data would otherwise be revealed). The constellation key points to the parent constellation 302 for a given fact table 304. The fact table name is the name of the fact table used in constructing the corresponding physical table names in the datamart 150. The truncate stage flag is used to indicate whether or not to truncate the fact staging table on the next extraction.

Detailed Description Text (139):
The fact column 310 lists all of the fact attributes within a single fact table 304. The fact column 310 includes a cleanse flag, a description, a fact aggregate operator, a fact column key, a fact column name, a fact column number, a fact table key, and a physical type. The fact aggregate operator is an SQL operator used to aggregate this fact column in the datamart 150. The fact column key is the primary key for the fact column. The fact column name is the physical name of the fact column. The fact column number counts and orders the number of columns in the fact table. The fact table key points to the fact table to which the corresponding fact column belongs. The fact table key points to the fact table to which the fact column belongs. The physical type is the database type for the fact column. This type is a logical type and provides for independence of implementation of the datamart 150 from the underlying database used.

Detailed Description Text (143):
The dimension base 306 is the metadata 160 describing all the dimension tables that can be used in a given constellation 302. These dimension bases can then be used in multiple constellations. The dimension base 306 includes the following attributes:

an aggregate key operator, a cleanse flag, a description, a dimension base key, dimension base name, a dimension base type, and a truncate stage flag. The aggregate key operator is an SQL operator used by the aggregate builder 170 to build aggregates from a dimension. The cleanse flag and description act similarly to those attributes in other tables. The dimension base key is the primary key for the dimension base 306. The dimension base name is the name of the base dimension used in constructing real tables in the datamart 150. The dimension base type indicates the type of a dimension base (either default or special (special includes "date" and "transaction type," which are used by the system 100). The truncate stage flag operates in the manner similar to other truncate stage flags.

Detailed Description Text (144):
The dimension column 329 defines the list of dimension attributes that are valid for a single base dimension 306 and inherited by a dimension usage. The dimension column 329 includes a cleanse label, a cleanse map key, a cleanse type, a description, a dimension base key, a dimension column key, a dimension column name, a dimension column number, a dimension number key, grouped by field, a physical type, a primary key, a time navigation field, and a default value. The cleanse label is a label presented to users after this column has been cleansed. The cleanse map key is for use when cleansing using value mapping. The cleanse map key indicates the mapping group to use. The cleanse type is the method for cleansing the dimension column 329. The description is for documenting the dimension column 329. The dimension base key is the numbered base in which the column resides. The dimension column key is the primary key for the dimension column 329. The dimension column name is the physical name of the column. The dimension column number is the count of the dimension columns (to prevent too many from being created in the datamart 150). The dimension node key is the aggregate hierarchy group in which the column resides. The "group by" field is used for special dimensions to indicate whether or not this column needs to be "grouped by" during the processing by the aggregate builder 170. The physical type is a logical database type for this dimension column 329. The primary key is used in special dimensions to indicate whether or not this column is the primary key. The time navigation field is for the date special dimension to indicate whether or not time navigation should use this field. The default value is the default value for the dimension column.

Detailed Description Text (146):
The dimension role 320 is a metadata 160 table that describes all of the dimension tables used in a constellation 302. The dimension role 320 includes a constellation key, a degenerative number, a description, a dimension base key, a dimension role key, a dimension role name, and a dimension role number. The constellation key points to the constellation 302 in which the dimension role 320 resides. The degenerative number defines the order of degenerate columns within fact tables in a constellation. The description is a documentation field for describing a dimension role. The dimension base key is the dimension base that this dimension role refers to. The dimension role key is the primary key for the dimension role 320. The dimension role name is the name of the dimension role and is used when constructing the foreign keys in the fact tables in the datamart 150. The dimension role number defines the order of the dimension roles within a constellation. That is, a constellation may have multiple dimension roles and the dimension role number allows for an ordering of those dimension roles.

Detailed Description Text (151):
The semantic instance 308 is a single record that represents the manner in which a fact or dimension table is extracted from staging tables, manipulated, and then used to populate the corresponding table in the datamart 150. The semantic instance 308 includes an extraction node key, dimension base key, a fact table key, a semantic instance key, and a semantic type key. The extraction node key points to the extraction node that a particular semantic instance belongs to. The dimension base key is the dimension base table owning this semantic instance. The fact table key points to the fact table owning this semantic instance. Only one of the

dimension base key and the fact table key is filled in for a semantic instance 308 because the semantic instance can only be applied to one or the other. The semantic instance key is a primary key for the semantic instance 308. The semantic type key is the indicator of the type of transformation necessary to construct this type of semantic instance in the datamart 150.

Detailed Description Text (154):
The aggregate group 342 defines a set of aggregates to be built for a constellation. An aggregate group 342 will cause a combinatorial creation of many aggregate tables in the datamart 150. The consultant defines for which dimensions aggregates are to be built (e.g., the consultant will define that one, none, all, etc. columns of a dimension are to be aggregated on in an aggregate group). The aggregate filtering done by the query and reporting program 104 will select the most appropriate aggregates for a given query.

Detailed Description Text (160):
A special dimension base 391 provides details about special built-in dimensions in the system 100. The special dimension base includes an "always include an aggregate" field, a default aggregate dimension type, a dimension base key, a list order in fact, a physical type of key, an index flag, and a special dimension base key. The "always include an aggregate" field indicates whether or not this dimension table must always be included in all aggregates. The default aggregate dimension type is the default manner in which this dimension is included in aggregate groups. The dimension base key is the one to one relationship to a dimension base. The list order in fact is the order in fact tables that the foreign key to this table will be listed. The physical type of key is the logical database type that foreign keys in the fact tables that point to this special dimension will be. The index flag is used in indexing. The special dimension base key is the primary key for the special dimension base 391.

Detailed Description Text (162):
The physical type 330 defines a look up table of logical data types that are relational database management system (RDBMS) independent. The physical type 330 is a logical data type that works across various source systems storage types. The physical type 330 includes a database physical type, a default value, and a special type.

Detailed Description Text (168):
The metacolumn 334 is a column that occurs by default in tables in the datamart 150. The metacolumn 334 includes an actual table type, a list order, a metacolumn key, a metacolumn name, and a physical type. The actual table type indicates the type of physical table in which this special column should appear. The list order is the order this column occurs in tables of the appropriate type. The metacolumn key is the primary key. The metacolumn name is the physical name of the column when it is used. The physical type is the logical data type for this column.

Detailed Description Text (169):
The actual table type 336 is a look up table for actual table types. Actual table types can be fact, dimension stage, fact stage, dimension map, or dimension.

Detailed Description Text (170):
The aggregate group 342, the aggregate fact 340, the aggregate dimension set 372, the dimension column set 370, the dimension column set definition 374, the fact index 380, the fact index definition 384, and the fact index number 382 are for future use and are therefore optional. Each of these tables provides greater flexibility when defining the metadata 160, improves the performance of the system 100, or may otherwise enhances the system 100.

Detailed Description Text (171):
Top Level Metadata Use

Detailed Description Text (173):
It is important to note that many of the tables in FIG. 3 are actually used in providing layers of abstraction to allow for the reuse of information and non-abstract tables. Therefore, a consultant will often only deal with only some of the tables in the FIG. 3. For the purposes of describing how the metadata 160 can be used to define a schema for the datamart 150, these grouping and levels of abstraction tables will be described where appropriate.

Detailed Description Text (174):
Generally, a consultant will create a new datamart 150 by defining instances of the dimension bases 306, and constellations 302. Each instance corresponds to a row in the dimensions bases 306 table or the constellation 302 table. The constellation instances are defined by defining aggregates, dimensions, facts, measures, and ticksheets. The following describes the definition of a schema using the metadata 160. This corresponds to block 210 of FIG. 2.

Detailed Description Text (175):
Beginning with the facts in a constellation, the consultant defines a fact table 304 row that will define the hub table in a star schema supported by the constellation. Again, it is important to remember that the fact tables in FIG. 3 are for definitional purposes, and are not the real fact tables in the datamart 150. A row in the fact column 310 holds the details of what columns will be created for place holders of actual values in a corresponding fact table. Thus, for each fact, the consultant defines the various fact columns.

Detailed Description Text (177):
Remember that the dimension base 306 holds the information to define the actual dimensions of the tables in the datamart 150. The dimension role 320 allows for the reuse of the dimension base tables. Thus, different dimension roles can refer to the same dimension base. This provides an important feature of some embodiments of the invention where the same dimension bases can be used in multiple constellations or within the same constellation. The dimension columns 329 define the columns on which queries can be performed in the datamart 150. The dimension node table 326 helps relate the dimension columns 329. Thus, the consultant will have defined the basic schema for the datamart 150.

Detailed Description Text (178):
The aggregate group 342 defines how particular facts or dimensions are to be aggregated by the aggregate builder 170. These aggregated facts provide much faster queries in the datamart 150.

Detailed Description Text (179):
The cleansing map tables are for scrambling the data in the datamart 150 for presentations to people who want to see the functionality of the system 100, without having to reveal the actual data in the datamart 150.

Detailed Description Text (180):
The special dimensions are the transaction type table and date values that are included in every fact table. Because this is included in every fact table, the system 100 can rely on the existence of the transaction type during the various stages of datamart 150 creation, modification, querying, and the like.

Detailed Description Text (181):
Thus, the elements of FIG. 3 can be used to allow the consultant to define the schema definitions 161 for creating the tables in the datamart 150.

Detailed Description Text (182):
Extraction Metadata

Detailed Description Text (183):
The following describes the metadata 160 used in the extraction process 204. This metadata, represented as extraction schema 400, is shown in FIG. 4. The extraction process focuses around the job and connector tables. In general, these tables define the various steps in extracting the source system data into the staging tables 130 and performing the desired semantic conversions on that data.

Detailed Description Text (184):
Extraction Metadata List

Detailed Description Text (186):
Extraction Metadata Descriptions

Detailed Description Text (188):
The job 402 is a top level object for controlling the work flow during the extraction and loading process 204. The job 402 includes a check databases field, a check tables field, a description, a label, an initial load flag, a job key, a job name, a log file width, a mail to on error, a mail to on success, and a truncate flag. The check databases field indicates whether or not an attempt should be made to log into all the data stores before executing the job. The check tables flag indicates whether or not to check for the existence of all the tables in the datamart 150 before executing the job. The description is for documenting the job (usually done by the consultant). The enabled flag indicates whether or not a particular job can be run. The initial load flag indicates whether or not to ignore all previous time stamped constraints when running a particular job. The job key is the primary key for the job table 402. The job name is the internal name of the job. The log file width indicates how many characters wide to make rows in the log file output. The mail to on error, and the mail to on success indicate where E-mail messages should be sent after failure or success of the particular job. The truncate flag indicates whether or not to truncate any tables when running a job.

Detailed Description Text (195):
The connector time stamp 407 relates to information about incremental extraction. An incremental extraction is where increments of the data in the source system 110 are extracted. The connector time stamp includes a connector key, a connector time stamp key, current max date, a current max time stamp, a last max date, and a last max time stamp. The connector key points to the connector to which the connector time stamp applies. The connector time stamp key is a primary key. The current max date is an indicator of the proposed new system date of the last successful extraction. The current maximum time stamp is the proposed new SQL server time stamp field for the last successful extraction. The last maximum date is the system date of the last successful extraction. The last maximum time stamp is the SQL server time stamp field for the source system databases at the last successful extraction.

Detailed Description Text (197):
The connector column latch 409 defines information about incremental extraction based on a database column. The incremental extraction information is thus kept in the database and can be retrieved. The connector column latch 409 includes the following attributes: a column name, a connector column latch key, a connector key, a current maximum value, a last maximum value, and a table name. The table name is the name in the input data store for the corresponding connector. The column name is the column name within that table. The connector column latch key is the primary key. The connector key points to the connector to which this latch applies. The current max value represents the proposed new maximum value for the incremental extraction. This number is pushed into the last maximum value if the currently executing extraction succeeds. The last maximum value is the maximum value that was extracted during the last run of the extraction.

Detailed Description Text (201):

The extraction node 410 is a single node, or step, in the extraction tree. The extraction tree defines the order of extraction steps. An extraction node includes an extraction node type, a debug level, a debug level row, an enabled flag, an extraction group key, an extraction node key, a list order, and on error type, a parent extraction node key, and a phase. The extraction node type defines the type of extraction node (as defined in the extraction node type table 491). This relationship is important and allows for the conversion of data in the staging tables for use in the datamart 150. The debug level indicates how to debug a particular step during execution. The debug level row indicates which row to start debugging at for SQL statements. The enabled flag indicates whether or not to execute a particular SQL statement associated with the extraction node. The extraction group key points to, if not null, the name of the group. The extraction node key is a primary key. The list order and the phase define the order of the corresponding step within an extraction node's parent. The on error type indicates what to do if there is an error during the execution of the step associated with the extraction node. The parent extraction node key points to the parent extraction node of the present extraction node.

Detailed Description Text (207):
The external column 424 defines a column in a user defined extraction table. The external column 424 includes the attributes for documenting a particular external column, and the column name in the external table. A pointer to the external table, a list order of appearance in the external table, and a physical type of the logical database for this column are included in the external column 424.

Detailed Description Text (212):
The semantic type 430 defines a set of predetermined semantic types for use in defining a schema. The semantic type includes a logical name for a particular transformation. Associated with the semantic type are a dimension semantic type 432 and a fact semantic type 434. The dimension semantic type table 432 defines the ways in which dimension data in the staging tables 130 can be extracted and put into the datamart 150. Similarly, the fact semantic type defines the ways in which the information in the staging tables 130 can be put into the fact tables of the datamart 150. Both the fact semantic type 434 and the dimension semantic type 432 include pointers to an actual table type and are used to subset the full list of semantic types.

Detailed Description Text (214):
The adaptive template 438 is a semantic transformation template (e.g., an SQL program) that is used in the extraction of the data in the staging tables 130 to turn all source data into transactional data. The adaptive template 438 includes attributes indicating a logical name for an adaptive program used within semantic transformations.

Detailed Description Text (218):
The data store 440 defines a logical data source, or sink, used during the extraction. The data store 440 includes the following attributes: the data store key, a data store flag, a description, a name, a source system key, and a store type. The data store key is the primary key. The datamart flag indicates whether or not this data store is the special datamart store. Since the datamart 150 and the metadata 160 can reside in the same database or different, the data mark helps resolve the location of the datamart 150. The description is for documentation of the particular data store. The name is the logical name of the data store. The source system key points to the source system identifier to which this data store belongs. This allows live, and backup, source systems to share the same identifier. The store type indicates the store type of this data store.

Detailed Description Text (219):
The source system 442 is a logical identifier for a source system 110 from which data can be pulled. This allows two physical databases to act as one master

database and one backup, for example. The source system 442 includes a description attribute, a source system key and a source system name. The description is for documentation to describe the source system. The source system key is a primary key for this table. This number also becomes identified source system field in the staging tables 130 being filled. The source system names is a logical name for a source system 110 from which the system 100 is pulling data.

Detailed Description Text (222):
The Oracle store 454 defines information about particular Oracle databases. The Oracle store 454 includes the following attributes: a data store key, an instance name, an Oracle store key, a password, an SQL network name, a user name, and a version. The data store key is a one to one relationship key to the data store being defined. The Oracle store key is the primary key. The password and user name are used to access a specific Oracle system. The version number is the Oracle vendor version number. The SQL network name is the SQL net instance name. The store version is a version of the store type (the database vendor's version for example) that the system recognizes. The store version has a pointer to the store type being defined and also includes a version number attribute.

Detailed Description Text (223):
The SQL server store table 456 defines details about an SQL server system. The SQL server store includes the following attributes: a data store key, a database name, a password, a server, and SQL server store key, a user name, and a version. The data store key is a one to one relationship to a data store entry. The database name is an SQL server database name ($$DEFAULT means the database in which this role resides). The password is the SQL server password. $$ DEFAULT again means the password currently logged into to read this data. The server is the SQL server name. The SQL server store key is the primary key. The user name is the SQL server user name. The version is the vendor's version number of this SQL server. $$DEFAULT means use the default value for the current database being used. For example, the database name means the database in which this role resides.

Detailed Description Text (224):
Extraction Metadata Use

Detailed Description Text (225):
The following describes the tables of FIG. 4 in the context of the extraction process 204. The job, the job step, and the connector, group extraction steps for extracting information from the source systems 110 and cause that information to be placed in the datamart 150. This organization allows for a very flexible extraction process. For example, where a two phase extraction is required, one connector could be used to extract the information from the source system 110, while a second connector could then be used to take this extracted data from an external table.

Detailed Description Text (227):
The following is an example illustrating the organization of job. Assume that a consultant wants to extract information from a source system that provided a raw set of home addresses. A system call could be run as part of a job step. The system call would determine the zip codes associated with those addresses. The zip codes could then be included in the datamart 150.

Detailed Description Text (230):
An SQL statement is a single step in an extraction run that represents a data push or a data pull. The SQL source code dictates the action for a given extraction node. After the SQL statements are run, the staging tables 130 are ready. The semantic conversion of the data in the staging tables 130 can occur.

Detailed Description Text (231):
The semantic instance represents the use of a single generic template on one fact or dimension table. The semantic type associated with the semantic instance is one

of a number of pre-defined recognized data meanings within the system 100 (e.g., an "order"). The semantic types correspond to programs for converting the data in the staging tables 130 into data for use in the datamart 150. An example of a semantic type is a "slowly changing dimension" type.

Detailed Description Text (232):
The semantics types, as mention previously, are made up of a series of templates. These templates include tokens that can be replaced with information from the corresponding dimension base or fact table. An example of an adaptive template is one that would be used in re-indexing of a fact table. This could be used as the last step in the semantic transformation of facts. The re-indexing will help speed the operation of the datamart 150. Importantly, this same indexing is performed for each fact table. No matter which semantic type is chosen for a given fact table, the same indexing is performed. Thus, this adaptive template can be used in each semantic type through its semantic type definition.

Detailed Description Text (239):
In some embodiments, the pre-parsed templates include additional tokens to deal with specific data stores. For example, the "select into" statement is a token in the pre-parsed version. This compensates for whether the data store is in Oracle database or an SQL server.

Detailed Description Text (243):
As mentioned previously, the use of the semantic types significantly reduces the amount of work needed to implement the datamart 150. By selecting a semantic type for a particular fact table or dimension table, the consultant automatically selects the corresponding pre-parsed SQL adaptive templates. The selected adaptive templates are then automatically converted into post-parsed SQL statements that include the schema specific information for the datamart 150. Additionally, these post-parsed SQL statements include the SQL for converting the data in the staging tables 130 into data that can be used in the datamart 150 tables.

Detailed Description Text (246):
The team template is used to properly populate an "associative" dimension table. Such a table is used whenever there is a one-to-many relationship between an individual fact row and a dimension. For example, if multiple salespeople can split the credit for an order, one needs some way to represent this situation in the datamart. In a star schema, one normally associates a tuple of dimension values with a fact row (e.g. product, customer, salesrep dimensions for the fact row containing price, quantity etc.). Since there is only a single salesrep_key, one could normally have only one salesrep associated with this transaction. There are two solutions. One is to introduce multiple fact rows for a transaction involving one to many relationships. If there were three salesreps on a specific order, there would be three fact rows for this order stored in the database. This has the disadvantage of multiplying the data size by a factor of three and slows queries. Also queries that are concerned with the total number of transactions become more difficult to process since duplicate rows, due to the multiplication by the number of salesreps, must be eliminated.

Detailed Description Text (247):
Another solution is to introduce an associative table between the actual salesrep dimension table and the fact table. Conceptually, the associative table contains "teams" of salespeople. If salesreps A, B and C often sell products together, they will be associated with a unique team key. The team key will be stored in each fact row for orders sold by the A, B, C team. The associative table will associate the team key with the three rows for A, B and C in the salesrep table. The associative table will have 3 rows representing this team (A-key, team1-key), (B-key, team1-key) and (C-key, team1-key). If the team of A, B, D and Q also sold products together, the associative table would have four additional rows (A-key, team2-key), (B-key, team2-key), (D-key, team2-key), (Q-key, team2-key). The team template scans

the staging table used to load the fact table and generates the appropriate rows for entry into the associative table, only for those teams THAT ACTUALLY OCCUR in the fact rows being loaded.

Detailed Description Text (256):
The population of both the denormalized team dimension and the associative table are difficult to code properly. This is especially true if this is done incrementally (e.g., on nightly extracts) and if you want to be independent of team order (e.g. A, B, C) is the same as (A, C, B). Thus, allowing the consultant to simply select this data semantic provides a significant improvement over previous systems.

Detailed Description Text (257):
Runtime Metadata

Detailed Description Text (258):
FIG. 5 illustrates the schema for the runtime environment within the system 100. The runtime schema 500 represents the schema description for the schema of the running datamart 150. That is, when the datamart 150 is created or modified, the schema definition is propagated into the runtime schema 500. Thus, the runtime schema 500 allows for the datamart 150 to be changed without having to rebuild all the tables and repopulate all of those tables. Additionally, the runtime metadata 500 provides the support for aggregate navigation. Aggregate navigation involves determining which aggregate to use in response to a query. Schema modification and aggregate navigation will now be explained in greater detail.

Detailed Description Text (259):
The schema modification involves comparing the changed schema definition with the present schema definition. As will be seen below, an actual table 502 keeps track of all of the dimension tables and the fact tables in the datamart 150. When a change is made to the schema definition, a comparison is made between the old definition and the new definition. The difference between these definitions defines the set of tables, columns, and rows that need to be added, deleted or modified, in some way. Importantly, the modifications can often be made without losing any data in the datamart 150.

Detailed Description Text (260):
The aggregate navigation process determines which aggregate most closely suits a particular query. The runtime metadata 160 keeps track of the aggregates available in the datamart 150. The query and reporting program 104 initiates a view of the runtime metadata 500 (in particular, the tables holding the aggregate tables definitions). The view results indicate which aggregates are available to answer the particular query. The view results are further examined to determine the best aggregate to use (the one that most closely corresponds to the query).

Detailed Description Text (263):
Runtime Metadata List

Detailed Description Text (264):
The runtime schema 500 includes the following elements: an actual table 502, an actual column 504, a fact aggregate table 512, a fact aggregate dimension 514, a dimension base aggregate 516, a dimension base aggregate column 518, a datamart letter 510, the dimension base 306, the fact table 304, the external table 422, an actual column 504, a physical type definition 530, an actual table type 336, an actual column type 540, the physical type 330, a database physical type 595, the translation string 332, a translation actual 539, a store type 450, a date 560, a business process 570, an adaptive template profile 580, and a transaction type 590.

Detailed Description Text (265):

Runtime Metadata Descriptions

Detailed Description Text (266):
The actual table 502 corresponds to metadata 160 that describe which dimension base and fact tables "actually" exist in the datamart 150.

Detailed Description Text (267):
The actual table 502 includes the following attributes: an actual table key, an actual table name, an actual table type, a dimension base key, an external table key, a fact table key, an index flag, a mirror flag, and a logical table name. The primary key is the actual table key. The actual table name corresponds to the physical name of this table in the database implementing the datamart 150. The actual table type is the logical type of this physical table. For example, if this is a dimension staging table or a fact staging table. The dimension base key points to the dimension base table definition that defined the corresponding physical table. The external table key points to the external table definition that defined the physical table. The fact table key points to the fact table definition that defined the corresponding physical table. The index flag and the mirror flag are used in indexing and mirroring, respectively. The logical table name defines the logical name for this table.

Detailed Description Text (268):
The actual column 504 is metadata describing a physical column in a physical table in the datamart 150. The actual column table latches this definition information when the physical tables are built in the datamart 150. The actual column 504 includes the following attributes: the actual column key, an actual column name, an actual column type, an actual table key, a dimension role name, a foreign table key, a group by field, a hierarchy, a list order, a parent hierarchy, a physical type, a primary key, and a time navigation field. The actual column name is the name of the physical column in the physical table in the datamart 150. The actual column type is the logical type of the column. The actual table key points to the actual table in which the actual column lives. The dimension role name is the logical role name of the dimension in the fact table for dimension foreign keys inside of a fact table. The foreign table key points to the actual dimension base tables in the actual tables 502 (the foreign table key is applicable to fact actual columns that are foreign keys to dimensions). The group by field, for dimension table, is true when this column should be included in an aggregate builder group. The hierarchy for dimension, for dimension columns, indicates that aggregate builder group to which this column belongs. The list order is the order of the column in the actual table 502. The parent hierarchy, for dimension columns, indicates the parent aggregate builder group to which this column belongs. The physical type is a logical data type of the column. The primary key, for dimension tables, is true when this column is the primary key of the actual table 502. The time navigation field, for the database dimension, is true if this field can be used by the time navigator.

Detailed Description Text (269):
The fact aggregate table 512 includes a list of fact aggregates in the datamart 150. The fact aggregates includes attributes that point to the actual fact table in which this aggregate belongs. The fact aggregate table 512 indicates which numbered aggregate represents the fact table in question, the number of rows in this aggregate, a datamart letter, and an enabled flag. The datamart letter indicates the mirrored datamart to which this fact aggregate belongs.

Detailed Description Text (270):
Mirror is used to ensure that partially completed extractions from the source systems 110 do not cause the database to become inconsistent.

Detailed Description Text (272):
The dimension base aggregate table 516 lists all the dimension aggregates in the

datamart. The dimension base aggregate includes the following attributes: an actual table key, an aggregate number, an aggregate size, a datamart letter, a dimension base aggregate key, and an enable flag. The actual table key points to the physical header for this dimension base. The aggregate number, for the dimension table in question, is the number of this particular aggregate. The aggregate size is the number of rows in the aggregate. The datamart letter indicates which mirrored database this aggregate lives in. The dimension base aggregate key is the primary key. The enable flag indicates whether or not the aggregate navigator should work with this aggregate.

Detailed Description Text (274):
The datamart letter 510 indicates which of two mirrored datamarts a particular aggregate belongs to. This is an optional element which may not be required if mirroring does not occur in the datamart 150. Mirroring duplicates the tables in the datamart 150. Changes can then be made to one copy of the datamart 150, while the other datamart 150 continues running. These changes can then be propagated when possible.

Detailed Description Text (275):
The actual column type 540 is a logical description of role a column plays in the system 100. The actual column type 540 includes attributes that define the default value to be used in a database for a column of this type.

Detailed Description Text (277):
The database physical type 595 defines the name of the physical database.

Detailed Description Text (278):
The translation actual table 539 defines the actual values of translations strings for a single relational database management system. These translations strings are the real strings to use for a given translation string within a store type. The translation actual table 539 also includes attributes that point to the store type.

Detailed Description Text (280):
The date table 560 is used to track date information in the datamart 150. Importantly, times and dates are always treated corrected in the datamart 150. This can be guaranteed because the consultant cannot change the definition of dates in the datamart 150. Thus, for example, the month of September will always have 30 days, and leap years will be handled correctly.

Detailed Description Text (283):
The business process table 570 is a look up table for supported business process types during the extraction. The business process 570 includes a business process key and a process name. The-process name corresponds to a logical name for a business process to which fact staging table belongs. The process key identifies a business process record in a fact staging table.

Detailed Description Text (286):
The traditional solution in datamarts is to store periodic "snapshots" of the balance. The snapshots are often stored at daily intervals for the recent historical past, and at greater intervals (e.g. weekly or monthly) for less recent history. This approach has two big disadvantages. The first is an enormous multiplication of data volume. If, for example, you are keeping track of inventory in a store you must store a snapshot for each product you hold in inventory for each day, even if you only sell a small fraction of all of your products on a given day. If you sell 10,000 different products but you only have 500 transactions a day, the "snapshot" datamart is 20 times larger than the transactional datamart. The second disadvantage relates to the most common solution for alleviating the first problem, namely storing snapshots at less frequent intervals for less recent history. This results in the inability to compare levels of inventory in

corresponding time periods since the same level of detail is not present in earlier data. For example, in manufacturing companies it is often the case that much business is done near the end of fiscal quarters. If one wants to compare inventory levels between Q1 1995, Q1 1996 and Q1 1997, and focus on the most important changes which occur near quarter end, one cannot use the approach of storing the snapshots at coarser levels of detail since daily data would be required.

Detailed Description Text (287):
In some embodiments of the system, the aggregate tables are used to answer queries about backlog/balance/inventory quantities. In order to answer such queries, the previously described rolling forward from the beginning of time is done. However, this is performed efficiently through the accessing of the appropriate time aggregates. For example, assume the datamart 150 has five years of historical transaction data beginning in 1993. Assume that one desires the inventory of some specified products on May 10, 1996. This would be computed by querying all of the transactions in the 1993, 1994 and 1995 year aggregates, the 1996 Q1 quarter aggregate, the April 1996 month aggregate, the May 1996 week 1 aggregate and finally 3 days of actual May, 1996 daily transactions. These transactions (additions and subtractions from inventory) would be added to the known starting inventory in order to produce the inventory on May 10. Note that this "time navigation "hops" by successively smaller time intervals (year, quarter, month, week, day) in order to minimize the number of database accesses. What is important is the exploitation of aggregate tables, that already exist in the system in order to answer transactional queries rapidly (e.g. What were the total sales of product X in April 1996?). This avoids the need to build what is essentially a second data datamart with the balance/inventory/backlog snapshots.

Detailed Description Text (288):
Query Mechanism Metadata

Detailed Description Text (289):
The following describes the metadata 160 used in the query/reporting program 104. This metadata is shown in FIG. 6. Generally, the query mechanism metadata can be broken into ticksheet metadata, measurement metadata, filtering metadata and display options metadata. The ticksheet metadata defines the user interface objects for user interaction with the datamart 150. The ticksheet defines how users can initiate queries and how results are presented back to the user. The measurement metadata defines a logical business calculation that can be presented to a user. Typically, the measurement metadata defines a format for presenting information to user that is more easily understood by the user or provides a more valuable result to the user. The filtering metadata defines how a user can filter results. Filtering allows the results set to be limited to particular dimension values. The display options metadata defines display options that can be provided to the user.

Detailed Description Text (290):
The following describes some important features of the user interface. The user interface allows the user to drill down through data. Also, portions of the query forms can be dynamic based upon values in fields (e.g., a list box can be dynamically updated because it is tied to a field in the datamart 150, that when changed, cause the values in the list box to change). Also, a query is guaranteed to be consistent with the schema because the query is tied to the schema definition.

Detailed Description Text (293):
Query Mechanism Schema Metadata Descriptions

Detailed Description Text (294):
Ticksheets Metadata

Detailed Description Text (297):

The ticksheet column 608 defines a single column for displaying measure choices on a ticksheet. The ticksheet column table 608 includes the ticksheet column key, the list order, the ticksheet key, and the description. These columns and the ticksheet column table 608 operate in a manner similar to such columns in other tables in this metadata.

Detailed Description Text (300):
The attribute table 610 defines the dimension attribute choices within a ticksheet. These choices are tied to a single dimension column in the schema definition of the datamart 150. The attribute table 610 includes an abbreviation, an attribute key, a dictionary key, a dimension column key, a dimension role key, a hyperlink, a label, a list order, a name, and a ticksheet key. The abbreviation is the shortened user string for the attribute. The attribute key is the primary key for this attribute. The dictionary key is a pointer to the dictionary 640 that includes help message for a particular attribute. The dimension column key is the dimension column in which this attribute refers. For degenerate dimensions this reference is null. The dimension usage, within a constellation, is defined by the dimension role key. The hyperlink is an html text for navigating return values for this attribute to other web sites, such as a company name look-ups etc. The label is what the user sees for a particular attribute. The list order defines a sort order on pop-up menus where one is the topmost in the list. The name is the internal name for the attribute. The ticksheet key indicates the ticksheet to which this attribute belongs.

Detailed Description Text (305):
Measurement Metadata

Detailed Description Text (306):
The following describes the measures used in the query mechanism schema 600. The measure table 620 defines a top level object for a logical business calculation. The measure table 620 includes a constellation key, a description, a measure key, a measure unit, and a name. The constellation key points to the constellation in which the measure resides. The description is for documentation purposes. The measure key is the primary key for the measure table 620. The measure unit is an indicator of the manner in which numbers are to be displayed. The name is the logical name of the measure.

Detailed Description Text (310):
Filtering Metadata

Detailed Description Text (312):
The following describes the filtering tables. The filter block 650 is a top level grouping table for filter area within a ticksheet. The filter block 650 is tied to a particular dimension column in the schema definition. The filter block 650 includes, columns, a description, a dictionary key, a dimension column key, a dimension role key, a filter block key, a filter block type, a label, a list order, a name, a plural, a mapping flag, and a ticksheet key. The columns field indicates the number of columns in this filter block. The description is for documentation. The dictionary key points to the help dictionary. The dimension column key points to the actual column name and the datamart to be filtered on. A null value here means degenerate dimension as determined by the dimension role key. The dimension role key points to the dimension role in the constellation of the ticksheet that is the form key to filter on for all facts in this constellation. A null value here means that a special dimension shared by all constellations is being used. The filter block key is the primary key for this table. The filter block type points to the filter block type table 652 which defines the ways in which this filter block is displayed to the user (e.g., a checkbox or a radio button). The label is the text that appears to the user for the filter block. The list order is the order that the filter block should appear in a list. The name is the name of the filter block. The plural field is the text that appears to the user for the filter block. The mapping flag is used in mapping. The ticksheet key points to the ticksheet that

this filter block belongs.

Detailed Description Text (314):
The filter element table 656 defines individual values for a dimension attribute within a filter block. The filter element table 656 includes a dictionary key, a filter element key, a filter group key, a label, a list order, a name, an SQL statement, and a value. The dictionary key points to the user help text for a particular filter element. The filter element key is the primary key. The filter group key points to the filter group to which this element belongs. The label is the user displayed string for the element. A list order is the order of this element within a filter group. The name is the hidden name of this element. The SQL statement is an SQL statement used to build the list of values for a dynamic list box filter group. The value is the database value that this element translates into in a SQL "WHERE" clause.

Detailed Description Text (315):
Display Options Metadata.

Detailed Description Text (321):
User Interface Example of Defining Metadata

Detailed Description Text (323):
The following describes a constellation used in a business. In this example a new dimension is added very simply and the changes are automatically propagated into the datamart 150. The enterprise manager interface 192 is used by the consultant to define and manipulate the system 100.

Detailed Description Text (324):
FIG. 7 illustrates the enterprise manager interface 192. Multiple system 100's can be connected to through that interface. Many of the objects and tables in the system 100 are shown. The base dimensions definitions 710 correspond to the base dimensions available under the "epitest" datamart. The constellations 712 for this datamart include an expense constellation and a sales constellation 720. Thus, the sales constellation 720 would appear as a row in the constellation table 302. Under the sales constellation 720 appear the definitions for the sales aggregates 721, the sales dimensions 723, the sales degenerate dimensions 725, the sales facts 726, the sales measures 728, and the sales ticksheets 729. Also, the extraction definitions 740 and security definitions for the "epitest" datamart are accessible. The sales dimensions 723 define rows in the dimension role table 320. These rows include customer billed to, product, application, program, customer ship to, and territory.

Detailed Description Text (333):
FIG. 16 illustrates the results of a request by the consultant to generate the datamart 150 from the definitions of the datamart. The results show that a number of tables have been created in the datamart 150. Importantly, FIG. 16 illustrates the results of an initial build process. In subsequent modifications, only those elements of the datamart that have changed will be changed. In other words, the subsequent changes are handled as an update process. An example of the update process is described below.

Detailed Description Text (335):
The following describes the creation of the connectors 162. Once the schema definitions 161 are set, the consultant then defines the connectors 162 to the source systems 110. The connectors, as noted above, define how information is to be extracted from the source systems 110 and how that information is to be placed into the datamart 150.

Detailed Description Text (338):
FIG. 19 illustrates the All Semantics connector as defined in the connector

definition window 1900. This connector includes the description and a definition of the input and output data stores. In this case, both of the data stores are the "epimart" (which is the datamart 150).

Detailed Description Text (340):
Returning to the discussion of connectors, FIG. 21 illustrates the connector entitled MFG. The MFG connector has two major steps: (1) order dimension staging, and (2) order fact staging. The results of these extraction steps are put in the staging tables 130. (The all extraction steps window 2100 illustrates all the possible steps in the system that can be used.)

Detailed Description Text (341):
FIG. 22 illustrates the SQL statement window 2200. The SQL statement window 2200 has an SQL field 2210 that includes the SQL statements that loads a customer table. As shown in the dialog box, the table references for the SQL statement includes the customer dimension table column definitions. That is, this SQL statement is going to be used to populate the customer dimension table.

Detailed Description Text (342):
In this example, the base name, type code, type name, region code, region name and tier name corresponds to the column names within the customer dimension. The date modify is an additional field that is to be used to indicate when this field was last modified in the database. Additionally, there is a source system key that is automatically included in every dimension. The source system key helps ensure that the datamart 150 is well-formed.

Detailed Description Text (344):
FIG. 23 illustrates the SQL statement for the Open Order Stage for populating the order fact table.

Detailed Description Text (345):
At this point the steps for generating the staging table information are complete. Now the semantic conversion steps are defined.

Detailed Description Text (346):
In FIG. 24, returning to the connector steps window, we have switched to an All Semantics connector 2410. The All Semantics connector 2410 causes the semantic conversion of the information in the staging table for use in the datamart 150.

Detailed Description Text (349):
FIG. 27 illustrates the results of a consultant adding a new dimension 2700 (called warehouse) to the sales constellation 720. The batch operation window 1600 illustrates the changes that are being made to the datamart that was created in FIG. 16. To achieve these results, the consultant need only perform the following steps:

Detailed Description Text (351):
2. Define the connector steps, including the SQL Statement to extract the warehouse data from the source systems 110.

Detailed Description Text (354):
5. Have the enterprise manager 102 update the datamart 150.

Detailed Description Text (355):
Thus, changing the schema definition of the datamart 150 is significantly simpler than previous systems.

Detailed Description Text (364):
FIG. 34 illustrates another query form 3200 generated from a different ticksheet definition. When the user selects the create report button, the query is issued

against the datamart 150. FIG. 35 illustrates some sample results from such a query.

Detailed Description Text (365):
The following query log illustrates the actual query that was executed against the datamart 150. The query log illustrates that an aggregate and navigation process determined which aggregate would be the most appropriate. The aggregate builder had created these aggregates. The most appropriate aggregate for the requested query was selected. The results were then returned.

Detailed Description Text (369):
Importantly, various embodiments of the invention do not necessarily include all of the features described above. For example, some embodiments of the invention do not include the first phase of the extraction process (loading the staging tables) because the source system data is provided to the system 100 directly by other extraction programs. Another example is where the datamart 150 is created, but a separate query interface is used to query the datamart 150. The query interface could use only a different communications protocol (e.g., instead of HTTP), or could be a completely different front end.

Detailed Description Text (371):
Some embodiments of the invention build a database system, not necessarily a datamart. Additionally, these embodiments do not have to conform to a star schema definition in the metadata 160.

Detailed Description Text (372):
An object database system could be generated instead of a relational database system.

Detailed Description Text (373):
Some embodiments of the invention comprise only a computer readable media (e.g., a CD, a tape, a hard drive or other storage media) that has the programs that implement all, or a portion of, the system 100. Some embodiments of the invention include an electromagnetic waveform having the programs. Some embodiments of the invention include only the computer system running the datamart, other embodiments of the invention include only the computer system that creates, accesses, and queries the datamart, but does not include in the datamart itself

Detailed Description Paragraph Table (1):
```
************************************************************************* *****
Query log
************************************************************************* *****
time <A Date Here> addr : 192.0.0.210 host : 192.0.0.210 user : agent :
Mozilla/4.01 [en] (WinNT; U) Datebase information: DRIVER={SQL
SERVER};SERVER=bigfoot; DATABASE=macromedia Keys and values coming in from the
browser: file = fileDesc = queryaction = QUERY hidden_queryaction = QUERY
OK_callback = NOK_callback = ticksheet = Orders hidden_ticksheet = Orders Rows =
Customer hidden_Rows = Customer Columns = Fiscal Year hidden_Columns = Fiscal Year
units = Price hidden_units = Price facttype = Shipped hidden_facttype = Shipped
facttype2 = Gross hidden_facttype2 = Gross stage = Orders hidden_stage = Orders
currencyunits = Thousands hidden_currencyunits = Thousands rowtotal = yes
hidden_rowtotal = yes columntotal = yes hidden_columntotal = yes percent = none
hidden_percent = none precision = 0 hidden_precision = 0 charts = 3D hidden_charts
= 3D maxrows = 10 hidden_maxrows = 10 rowsorttype = value hidden_rowsorttype =
value Fiscal_Year = All hidden_Fiscal_Years = All Fiscal_Quarters = All
hidden_Fiscal_Quarters = All Calendar_Months = All hidden_Calendar_Months = All
Business_Units = All hidden_Business_Units = All Product_Lines = All
hidden_Product_Lines = All Product_Supergroups = All hidden_Product_Supergroups =
All Platforms = All hidden_Platforms = All Product_Languages = All
hidden_Product_Languages = All Product_SKUs = All hidden_Product_SKUs = All Product
```

```
SKU = Sales_Reps = All hidden_Sales_Reps = All Channels = All hidden_Channels = All
Customer_Types = All hidden_Customer_Types = All Customer_Regions = All
hidden_Customer_Regions = All Customers = All hidden_Customers = All Customer =
sqStyle = classic hidden_sqStyle = classic Contents of %FormData: Columns = Fiscal
Year rowsorttype = value ticksheet = Orders Customers = All charts = 3D Customer
Types = All Product SKUs = All Customer Regions = All Fiscal Quarters = All Rows =
Customer Channels = All precision = 0 Product Supergroups = All percent = none
maxrows = 10 Sales Reps = All columntotal = yes Calendar Months = All queryaction =
QUERY sqStyle = classic Fiscal Years = All Product Languages = All Platforms = All
Product Lines = All rowtotal = yes currencyunits = Thousands Business Units = All
The colheaders are: The Cellitems are: Price Shipped Gross Orders The Cellitems
abreviated are: Dollar Amount Shipped Gross Orders pid is: 310 spid is: 25 The
valid collheaders are: The invalid colheaders are: The valid cellitems are: Price
Shipped Gross Orders The invalid cellitems are: The unitstack is: CURRENCY The
cellstack is: -SUM (Order.net_price) The selectstack is: -SUM (Order.net_price) The
typestack is: SHIP Transtypes are: "BEGIN_RETURN" = 1007 "END_GROSS" = 1004
"END_SRBDTH" = 1018 "END_SRADJ" = 1012 "BOOK" = 1 "BEGIN_ANET" = 1013 "END_IADJ" =
1024 "END_ICOMP" = 1022 "BEGIN_GROSS" = 1003 "BEGIN_SBBDTH" = 1017 "END_SBDTH" =
1016 "END" = 1002 "BEGIN_SPADJ" = 1011 "END_SADJ" = 1010 "BEGIN_IADJ" = 1023
"BEGIN_ICOMP" = 1021 "END_NET" = 1006 "SHIP_ADJUST" = 103 "LOST" = 3 "END_SALL" =
1020 "BEGIN_SBDTM" = 1015 "BEGIN" = 1001 "BEGIN_SADJ" = 1009 "BOOK_RETURN" = 2
"END_FADJ" = 1026 "BEGIN_NET" = 1005 "BEGIN_SALL" = 1019 "GL" = 105 "SHIP_RETURN" =
102 "SHIP_RADJ" = 104 "SHIP" = 101 "END_RETURN" = 1008 "INV_ADJUST" = 201
"LEAD_LOST" = 4 "BEGIN_FADJ" = 1025 "FINV_ADJUST" = 202 "END_ANET" = 1014 The
facttable is: Order The limitvalues are: The access liinitations are: The
limitvalues are: The header took 261 milliseconds. Parsing took 160 milliseconds.
Generating the header took 0 milliseconds. Building user limits took 0
milliseconds. Phase 0: Aggregate navigator initialization. Phase 1: Getting
dimensions from SQL. ( 10 ) Phase 2: Getting fields available from SQL. ( 90 )
Phase 3: Getting degenerate fields from SQL. ( 0 ) Phase 0: Preparing for query
building and execution.R2[0 -> 0] = (SUM(Z0)) R1[0] = SUM(Z0) The column router:
Cell location 0 will be returned in column 0 when Type is SHIP. The result router:
Result location 0 is (SUM(Z0)) 0 The unique facttables are:Order The number of
unique facttables are: 1 The unique types are:SHIP Table to unique number lookup
Order => _0_ Begin work on the query based on the facttable Order Phase 1: Table
Order. Creating table aliases ( 10 ) JOIN_FIELD IS CUSTOMER_BILLTO_KEY FOR Customer
JOIN_FIELD IS FOR Fiscal Year SQL table aliases: Order.quadrature. : T3
Date.quadrature. : T2 Customer.quadrature.CUSTOMER_BILLTO_KEY : T1 Table aliases:
tablealiaslookup(T1) = Customer tablealiaslookup(T2) = Date tablealiaslookup(T3) =
Order selectalias Customer: T1.base_name Fiscal Year: T2.fy_name selectstackalias -
SUM(Order.net_price) : -SUM(T3.net_price) joinalias: Customer: T1.customer_key =
T3.customer_billto_key Phase 2: Building SELECT clause ( 0 ) Phase 3: Building FROM
clause ( 0 ) Phase 4: Building WHERE clause ( 0 ) Phase 5: Building GROUP BY clause
( 0 ) SQL before going through the aggregate navigator: SELECT Columns =
T2.fy_name, Type = T3.Transtype_key, C0 = -SUM(T3.net_price), Rows = Ti.base_name
INTO #tmp_0_ FROM Customer T1, Date T2, Order T3 WHERE T1.customer_key =
T3.customer_billto_key and T2.date_key = T3.date_key and T3.Transtype_key in (101)
GROUP BY T1.base_name, T2.fy_name, T3.Transtype_key
********************************************************************** *******
Selecting appropriate aggregate for the query.
********************************************************************** *******
Phase 0: Aggregate navigator. Preparing for query building and execution. Phase 1:
Spliting query into clauses. ( 0 ) Phase 2: Construction of aliases. ( 0 ) Phase 3:
Extracting neededfields from where clause. ( 0 ) Phase 4: Extracting neededfields
from group by clause. ( 0 ) Phase 5: Extracting neededfields from select clause.
( 0 ) Phase 6: Unaliasing. ( 0 ) Phase 7: Constructing the SQL to fetch smallest
aggregate. ( 20 ) Phase 8: Running the big SQL. ( 20 ) Phase 9: Extracting results
from the big SQL. ( 0 ) Phase 10: Adjusting input with aggregate information. ( 0 )
Phase 6: Aggregate Navigating ( 40 )
********************************************************************** ****
```

Appropriate aggregate determined (CUSTOMER_0, DATE_4, ORDER_86), now select
****

Other Reference Publication (1):
Kimball, R., "The Data Warehouse Toolkit", (1996) John-Wiley & Sons, Inc., 388 pages (includes CD ROM).

Other Reference Publication (6):
Kawaguchi, A. et al., "Concurrency Control Theory for Deferred Materialized Views", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 306-320.

Other Reference Publication (7):
Zhuge, Y. et al., "Consistency Algorithms for Multi-Source Warehouse View Maintenance", Distributed and Parallel Databases, vol. 6, pp. 7-40 (1998), Kluwer Academic Publishers.

Other Reference Publication (10):
Yang, J. et al., "Maintaining Temporal Views Over Non-Historical Information Sources For Data Warehousing", Advances in Database Technology--EDBT '98, Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, Mar. 1998, pp. 389-403.

Other Reference Publication (11):
Quass, D., "Maintenance Expressions for Views with Aggregation", Proceedings of the 21st International Conference on Very Large Data Bases, IEEE, Zurich, Switzerland, (Sep. 1995), 9 pages.

Other Reference Publication (12):
Mumick, I. et al., "Maintenance of Data Cubes and Summary Tables in a Warehouse", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 100-111.

Other Reference Publication (13):
Huyn, N., "Multiple-View Self-Maintenance in Data Warehousing Environments", Proceedings of the 23rd International Conference on Very Large Data Bases, IEEE, (1997), pp. 26-35.

Other Reference Publication (16):
Gupta, H., "Selection of Views to Materialize in a Data Warehouse", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 98-112.

Other Reference Publication (19):
Labio, W. et al., "Physical Database Design for Data Warehouses", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 277-288.

Other Reference Publication (23):
Tsuda, K. et al., "IconicBrowser: An Iconic Retrieval System for Object-Oriented Databases", Proc. of the IEEE Workshop on Visual Languages, Oct. 4, 1989, pp. 130-137.

CLAIMS:

1. A method of generating a datamart from a plurality of sources having a query mechanism interface using a computer system, the method comprising:

accessing a schema definition which describes a schema for the datamart, the schema defined using a number of semantic meanings describing transformations of data between the plurality of sources and the datamart;

accessing a description of the query mechanism interface to be generated in the datamart;

generating a set of commands from the schema definition, including,

generating a set of table creation commands, and

generating a set of table access and manipulation commands, the set of table access and manipulation commands corresponding to the semantic meaning of the schema; and

generating the query mechanism interface from the query mechanism description and the description of the schema.

6. The method of claim 1 wherein the query mechanism description defines a query form, the query mechanism description defining a set of columns to be displayed on the query form, each column of the set of columns corresponding to a queryable field, the set of columns allowing an user to define a query for the datamart.

8. A system for querying a datamart, the system comprising:

the datamart;

a schema definition defining the schema of the datamart;

a query interface description defining a user interface for querying the datamart, the query interface description being defined using the schema definition, the schema defined using a number of semantic meanings describing the schema in terms of the meanings of data; and

a first program for accessing the query interface description, generating a query form from the query interface description, receiving a query corresponding to the query form and issuing the query to the datamart.

Previous Doc Next Doc Go to Doc#

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

End of Result Set

☐ Generate Collection Print

```
  L6: Entry 13 of 13                    File: USPT                    Dec 12, 2000
```

US-PAT-NO: 6161103
DOCUMENT-IDENTIFIER: US 6161103 A

TITLE: Method and apparatus for creating aggregates for use in a datamart

DATE-ISSUED: December 12, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Rauer; Allon	Mountain View	CA		
Walsh; Gregory Vincent	Cupertino	CA		
McCaskey; John P.	Mountain View	CA		
Weissman; Craig David	Belmont	CA		
Rassen; Jeremy A.	Sunnyvale	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Epiphany, Inc.	San Mateo	CA			02

APPL-NO: 09/073733 [PALM]
DATE FILED: May 6, 1998

PARENT-CASE:
CROSS REFERENCES TO RELATED APPLICATIONS This application relates to the following group of applications. Each application in the group relates to, and incorporates by reference, each other application in the group. The invention of each application is assigned to the assignee of this invention. The group of applications includes the following. U.S. patent application Ser. No. 09/073,748, entitled "Method and Apparatus for Creating a Well-Formed Database System Using a Computer," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Eliot Leonard Wegbreit. U.S. patent application Ser. No. 09/073,752, entitled "Method and Apparatus for Creating and Populating a Datamart," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Lynn Randolph Slater, Jr. U.S. patent application Ser. No. 09/073,733, entitled "Method and Apparatus for Creating Aggregates for Use in a Datamart," filed May 6, 1998, and having inventors Allon Rauer, Gregory Vincent Walsh, John P. McCaskey, Craig David Weissman and Jeremy A. Rassen. U.S. patent application Ser. No. 09/073,753, entitled "Method and Apparatus for Creating a Datamart and for Creating a Query Structure for the Datamart," filed May 6, 1998, and having inventors Jeremy A. Rassen, Emile Litvak, abhi a. shelat, John P. McCaskey and Allon Rauer.

INT-CL-ISSUED: [07] G06F 17/30

INT-CL-CURRENT:

TYPE IPC DATE

CIPP G06 Q 30/00 20060101

US-CL-ISSUED: 707/4; 707/1, 707/3
US-CL-CURRENT: 707/4; 707/1, 707/3

FIELD-OF-CLASSIFICATION-SEARCH: 707/1-10, 707/200-208, 707/100-104
See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected | Search ALL | Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5386556	January 1995	Hedin et al.	707/4
☐	5550971	August 1996	Brunner et al.	707/3
☐	5659724	August 1997	Borgida et al.	707/3
☐	5675785	October 1997	Hall et al.	707/102
☐	5806060	September 1998	Borgida et al.	707/3
☐	5995958	November 1999	Xu	707/3

OTHER PUBLICATIONS

McAlpine, G. et al., "Integrated Information Retrieval in a Knowledge Worker Support System", Proc. of the Intl. Conf. on Research and Development in Information Retrieval (SIGIR), Cambridge, MA, Jun. 25-28, 1989, Conf. 12, pp. 48-57.
Tsuda, K. et al., "IconicBrowser: An Iconic Retrieval System for Object-Oriented Databases", Proc. of the IEEE Workshop on Visual Languages, Oct. 4, 1989, pp. 130-137.
"Multiple Selection List Presentation Aids Complex Search", IBM Technical Disclosure Bulletin, vol. 36, No. 10, Oct. 1993, pp. 317-318.
Kimball, R., "The Data Warehouse Toolkit", (1996) John-Wiley & Sons, Inc., 388 pages (includes CD ROM).
Chawathe, S. et al., "Change Detection in Hierarchically Structured Information", SIGMOD Record, vol. 25, No. 2, Jun. 1996, pp. 493-504.
Chawathe, S. et al., "Meaningful Change Detection in Structured Data", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 26-37.
Labio, W. et al., "Efficient Snapshot Differential Algorithms for Data Warehousing", Department of Computer Science, Stanford University, (1996), pp. 1-13.
Wiener, J. et al., "A System Prototype for Warehouse View Maintenance", The Workshop on Materialized Views, pp. 26-33, Montreal, Canada, Jun. 1996.
Kawaguchi, A. et al., "Concurrency Control Theory for Deferred Materialized Views", Database Theory-ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 306-320.
Zhuge, Y. et al., "Consistency Algorithms for Multi-Source Warehouse View Maintenance", Distributed and Parallel Databases, vol. 6, pp. 7-40 (1998), Kluwer Academic Publishers.
Zhuge, Y. et al., "View Maintenance in a Warehousing Environment", SIGMOD Record, vol. 24, No. 2, Jun. 1995, pp. 316-327.

Wisdom, J. "Research Problems in Data Warehousing", Proc. of 4th Int'l Conference on Information and Knowledge Management (CIKM), Nov. 1995, 6 pages.
Yang, J. et al., "Maintaining Temporal Views Over Non-Historical Information Sources For Data Warehousing", Advances in Database Technology--EDBT '98, Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, Mar. 1998, pp. 389-403.
Quass, D., "Maintenance Expressions for Views with Aggregation", Proceedings of the 21st International Conference on Very Large Data Bases, IEEE, Zurich, Switzerland, (Sep. 1995), 9 pages.
Mumick, I. et al., "Maintenance of Data Cubes and Summary Tables in a Warehouse", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 100-111.
Huyn, N., "Multiple-View Self-Maintenance in Data Warehousing Environments", Proceedings of the 23rd International Conference on Very Large Data Bases, IEEE, (1997), pp. 26-35.
Quass, D. et al., "Making Views Self-Maintainable for Data Warehousing", Proceedings of the Fourth International Conference, on Parallel and Distributes Information Systems, IEEE, Dec. 1996, pp. 158-169.
Gupta, H. "Selection of Views to Materialize in a Data Warehouse", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 98-112.
Harinarayan, V. et al., "Implementing Data Cubes Efficiently", SIGMOD Record, vol. 25, No. 2, Jun. 1996, pp. 205-216.
Gupta, H. et al., "Index Selection for OLAP", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 208-219.
Labio, W. et al., "Physical Database Design for Data Warehouses", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 277-288.
Gupta, A. et al., "Aggregate-Query Processing in Data Warehousing Environments", Proceedings of the 21st VLDB Conference, Zurich, Switzerland, Sep. 1995, pp. 358-369.
O'Neill, P. et al., "Improved Query Performance with Variant Indexes", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 38-49.

ART-UNIT: 271

PRIMARY-EXAMINER: Ho; Ruay Lian

ATTY-AGENT-FIRM: Wilson, Sonsini, Goodrich & Rosati

ABSTRACT:

A method for automatically defining aggregates for use in a datamart is described. The datamart includes fact and dimension tables. The method comprises accessing a schema description and an aggregates description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The aggregates description specifies the aggregates, which define, from the schema definition, which aggregate tables are to be created from the fact tables and dimension tables in the datamart. The data in the aggregates correspond to the pre-computed results of specific types of queries. In response to a query, the aggregates can be searched to determine an appropriate aggregate to use in response to that query. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create, populate and access, the aggregates are also created from the aggregates description. Some of the commands of the first set of commands are executed causing the creation and population of the tables. Some of the commands of the second set of commands are executed causing the creation of the aggregate tables. Some of the remaining commands of the second set of commands are executed to populate the aggregate

tables from the populated fact and dimension tables.

11 Claims, 43 Drawing figures

Previous Doc Next Doc Go to Doc#

L6: Entry 13 of 13 File: USPT Dec 12, 2000

DOCUMENT-IDENTIFIER: US 6161103 A
TITLE: Method and apparatus for creating aggregates for use in a datamart

Abstract Text (1):
A method for automatically defining aggregates for use in a datamart is described. The datamart includes fact and dimension tables. The method comprises accessing a schema description and an aggregates description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The aggregates description specifies the aggregates, which define, from the schema definition, which aggregate tables are to be created from the fact tables and dimension tables in the datamart. The data in the aggregates correspond to the pre-computed results of specific types of queries. In response to a query, the aggregates can be searched to determine an appropriate aggregate to use in response to that query. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create, populate and access, the aggregates are also created from the aggregates description. Some of the commands of the first set of commands are executed causing the creation and population of the tables. Some of the commands of the second set of commands are executed causing the creation of the aggregate tables. Some of the remaining commands of the second set of commands are executed to populate the aggregate tables from the populated fact and dimension tables.

Parent Case Text (3):
U.S. patent application Ser. No. 09/073,748, entitled "Method and Apparatus for Creating a Well-Formed Database System Using a Computer," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Eliot Leonard Wegbreit.

Parent Case Text (4):
U.S. patent application Ser. No. 09/073,752, entitled "Method and Apparatus for Creating and Populating a Datamart," filed May 6, 1998, and having inventors Craig David Weissman, Greg Vincent Walsh and Lynn Randolph Slater, Jr.

Parent Case Text (5):
U.S. patent application Ser. No. 09/073,733, entitled "Method and Apparatus for Creating Aggregates for Use in a Datamart," filed May 6, 1998, and having inventors Allon Rauer, Gregory Vincent Walsh, John P. McCaskey, Craig David Weissman and Jeremy A. Rassen.

Parent Case Text (6):
U.S. patent application Ser. No. 09/073,753, entitled "Method and Apparatus for Creating a Datamart and for Creating a Query Structure for the Datamart," filed May 6, 1998, and having inventors Jeremy A. Rassen, Emile Litvak, abhi a. shelat, John P. McCaskey and Allon Rauer.

Brief Summary Text (4):
This invention relates to the field of databases. In particular, the invention

relates to creating databases, and loading and accessing data in the databases.

Brief Summary Text (6):
Many different types of databases have been developed. On line transaction processing (OLTP) databases are examples of typical databases used today. OLTP databases are concerned with the transaction oriented processing of data. On line transaction processing is the process by which data is entered and retrieved from these databases. In these transaction-oriented databases, every transaction is guaranteed. Thus, at a very low level, the OLTP databases are very good at determining whether any specific transaction has occurred.

Brief Summary Text (7):
Another type of database is a data warehouse or datamart. A datamart transforms the raw data from the OLTP databases. The transformation supports queries at a much higher level than the OLTP atomic transaction queries. A data warehouse or a datamart typically provides not only the structure for storing the data extracted from the OLTP databases, but also query analysis and publication tools.

Brief Summary Text (8):
The advantage of datamarts is that users can quickly access data that is important to their business decision making. To meet this goal, datamarts should have the following characteristics. First, datamarts should be consistent in that they give the same results for the same search. The datamart should also be consistent in the use of terms to describe fields in the datamart. For example, "sales" has a specific definition, that when fetched from a database, provides a consistent answer. Datamarts should also be able to separate and combine every possible measure in the business. Many of these issues are discussed in the following book, Ralph Kimball, The Data Warehouse Toolkit, John Whiley and Sons, Inc., New York, N.Y. (1996).

Brief Summary Text (9):
Multi-dimensional datamarts are one kind of datamart. Multi-dimensional datamarts rely on a dimension modeling technique to define the schema for the datamart. Dimension modeling involves visualizing the data in the datamart as a multi-dimension data space (e.g., image the data as a cube). Each dimension of that space corresponds to a different way of looking at the data. Each point in the space, defined by the dimensions, contains measurements for a particular combination of dimensions. For example, a three dimensional cube might have product, customer, and territory dimensions. Any point in that cube, defined by those three dimensions, will represent data that relates those three dimensions.

Brief Summary Text (10):
The data in the datamart is organized according to a schema. In a dimensional datamart, the data is typically organized as a star schema. At the center of a standard star schema is a fact table that contains measure data. Radiating outward from the fact table, like the points of a star, are multiple dimension tables. Dimension tables contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are joined. The advantage of such a schema is that it supports a top down business approach to the definition of the schema.

Brief Summary Text (11):
Present datamarts have a number of drawbacks that are now discussed. First, datamarts are typically difficult to build and maintain. This is because of the requirements that they be consistent and flexible. A related drawback of present day datamarts is that they do not allow the consultants of the datamart to make changes to the schema simply and easily. Because datamarts support very high level queries about the business processes in the business, they require a great deal of

consistency in the use of data from the OLTP systems. Additionally, the datamarts need to be very flexible to address changes in the types of high level queries supported. Changing typical datamarts require the changing of hundreds, or potentially thousands, of lines of SQL code. For example, if a fact column is added to a fact table, the change propagates throughout the datamart. These changes are typically implemented by hand, a very time consuming and error prone process. As a result of the hand coding involved, it is quite possible to construct the database in an arbitrary fashion that does not conform to good rules for constructing datamarts. Thus, well-formed datamarts may not result.

Brief Summary Text (14):
One embodiment of the invention includes a method for automatically defining aggregates for use in a datamart. The datamart includes fact and dimension tables. The method comprises accessing a schema description and an aggregates description for the datamart. The schema description specifies a schema, which in turn, defines the relationships between the fact tables and dimension tables of the datamart. The aggregates description specifies the aggregates, which define, from the schema definition, which aggregate tables are to be created from the fact tables and dimension tables in the datamart. The data in the aggregates correspond to the pre-computed results of specific types of queries. In response to a query, the aggregates can be searched to determine an appropriate aggregate to use in response to that query. The schema description is used to create a first set of commands to create and populate the fact and dimension tables. Additionally, a second set of commands to create, populate and access, the aggregates are also created from the aggregates description. Some of the commands of the first set of commands are executed causing the creation and population of the tables. Some of the commands of the second set of commands are executed causing the creation of the aggregate tables. Some of the remaining commands of the second set of commands are executed to populate the aggregate tables from the populated fact and dimension tables.

Drawing Description Text (3):
FIG. 1 illustrates a datamart system representing one embodiment of the invention.

Drawing Description Text (4):
FIG. 2 illustrates an embodiment of a method of defining the datamart, loading the datamart, and then querying the data.

Drawing Description Text (5):
FIG. 3 illustrates a schema used in the system of FIG. 1 to define schemas for the datamart.

Drawing Description Text (9):
FIG. 7 through FIG. 29 describe a user interface that can be used to define a schema, build a datamart, load the datamart, and query the datamart.

Detailed Description Text (13):
DATAMART SYSTEM

Detailed Description Text (16):
Metadata Overview

Detailed Description Text (19):
EXAMPLE METHOD OF DEFINING AND USING THE DATAMART

Detailed Description Text (20):
TOP LEVEL METADATA SCHEMA

Detailed Description Text (21):
Top Level Metadata List

Detailed Description Text (22):
Top Level Metadata Descriptions

Detailed Description Text (30):
Top Level Metadata Us

Detailed Description Text (31):
EXTRACTION METADATA

Detailed Description Text (32):
Extraction Metadata List

Detailed Description Text (33):
Extraction Metadata Descriptors

Detailed Description Text (41):
Extraction Metadata Use

Detailed Description Text (45):
RUNTIME METADATA

Detailed Description Text (46):
Runtime Metadata List

Detailed Description Text (47):
Runtime Metadata Descriptions

Detailed Description Text (49):
QUERY MECHANISM METADATA

Detailed Description Text (51):
Query Mechanism Schema Metadata Descriptions

Detailed Description Text (52):
Ticksheets Metadata

Detailed Description Text (53):
Measurement Metadata

Detailed Description Text (54):
Filtering Metadata

Detailed Description Text (55):
Display Options Metadata

Detailed Description Text (56):
USER INTERFACE EXAMPLE OF DEFINING METADATA

Detailed Description Text (66):
The following describes a system according to various embodiments of the invention. Generally, the system allows a consultant to define a well-formed datamart. The system includes tables and columns that conform to the definition of the datamart. The system also includes additional columns for foreign key tracking, source system key mapping, time and date tracking. The system has automatic indexing. The system enforces typing information about the data stored in the datamart. These additional features cause the datamart to operate in a consistent manner. One benefit of such consistent operation is that results are consistent in meaning from query to query.

Detailed Description Text (67):

Focusing on the datamart creation, the system allows a consultant to build a datamart from a schema definition and a definition of the sources of the data. From the schema definition, the system automatically builds the tables needed in the datamart. Also, from the schema definition, and the sources definition, the system can automatically extract the data from those sources. Depending on the semantic meaning of the data, as defined by the schema definition, the system automatically converts the data from the sources into forms that are readily usable in the datamart. Once the datamart has been created, and the data has been loaded, users can then perform queries on the data.

Detailed Description Text (68):
As part of the datamart creation, the system allows the consultant to define aggregates for the datamart. The aggregates correspond to pre-computed query results for different types of queries. For example, an aggregate can be created for a query that asks for all sales, by region, by quarter. The corresponding aggregate table would include a set of rows that have the results for this query (e.g., each row includes the quarterly sales for each region). The aggregates are specified using the schema definition. This makes defining and changing aggregates relatively simple.

Detailed Description Text (69):
To allow a user to query the datamart, the system includes an interface for defining what fields can be used by the user to query the datamart. Additionally, by allowing the consultant to define measure and related information, the system allows the consultant to specify how the results are to appear to the users.

Detailed Description Text (70):
The following description first presents a system level view of primarily one embodiment. Then, an example use of the system is presented. Next, the metadata used in the system is described. This metadata description is broken into four parts: a top level description of the metadata used in defining schemas, a description of the metadata used during the extraction, a description of the metadata used while the datamart is running, and a description of the query interface metadata. Next, an example set of user interface screen shots illustrates how consultants can quickly and efficiently define schemas, aggregates, and query interfaces, and how users can query the datamart. Next, additional alternative embodiments are described.

Detailed Description Text (72):
Datamart or Data Warehouse--is a database.

Detailed Description Text (73):
Schema--is a description of the organization of data in a database. Often, the schema is defined using a data definition language provided by a database management system. More abstractly, the schema can be the logical definition of a data model for use in a database.

Detailed Description Text (74):
Metadata--is data that defines other data. This is not the actual data in the datamart, but is the data that defines the data in the datamart.

Detailed Description Text (75):
Constellation--a grouping of dimension definitions, fact definitions, like-structured facts (all facts in a constellation have the same dimensional foreign keys), or stars, and other metadata definitions. Often the grouping relates to a business process (e.g., sales).

Detailed Description Text (76):
Fact Table--the central table of a star schema. It stores the numeric measurements of the business that is supplying the information to the datamart.

Detailed Description Text (78):
Dimension--the tables that link to the fact table in a star schema. The tables store the descriptions of the dimensions of the business. Examples of dimensions are product and territory.

Detailed Description Text (80):
User--any end user who would normally wish to query a datamart, but would not usually be concerned with the implementation or maintenance of the datamart.

Detailed Description Text (81):
Consultant--is a person responsible for the creation and maintenance of a datamart.

Detailed Description Text (82):
Source System--is any computer system that holds the raw data used by the system. Examples of such source systems are OLTP database systems.

Detailed Description Text (83):
Data Store--any data storage (physical or logical) from which data is received or to which data is stored. Examples of a data store are files, a database, etc,

Detailed Description Text (86):
Datamart System

Detailed Description Text (87):
FIG. 1 illustrates a datamart system representing one embodiment of the invention. The system supports the creation of a well-formed datamart. This system allows consultants to use metadata to define schemas for a datamart. From the definition of the schema, the system can automatically generate the tables in the datamart. Further, the system can automatically extract the data from the source systems, perform conversions on that data and populate the datamart. The system supports the automatic creation and processing of aggregates from aggregate definitions. The system also supports the creation of the query mechanisms from query definitons.

Detailed Description Text (90):
FIG. 1 includes the following elements: source systems 110, a system 100, a web server 186, a consultant computer 190, and a user computer 180. The system 100 includes the metadata 160, an enterprise manager 102, an extraction program 120, staging tables 130, a semantic template conversion program 140, a datamart 150, an aggregate builder 170, and a query and reporting program 104. The metadata 160 includes the following data: schema definitions 161, connectors 162 (connectors are also referred to as extractors), semantic definitions 163, source system information 164, aggregate information 167, measurement information 168, and query/reporting information 169. The user computer 180 is shown running a browser 182. The browser 182 includes a query/results interface 184. The consultant computer 190 shows the enterprise manager interface 192 which shows the metadata organization of the system 100.

Detailed Description Text (92):
The following describes the metadata 160, then the other elements of the system 100, and finally, the elements that are external to the system 100. These elements are all described in greater detail below.

Detailed Description Text (93):
Metadata Overview

Detailed Description Text (94):
The metadata 160 includes many different types of data and information. This information can be broken down into information related to (1) the definition of

the schema for the datamart 150, (2) the data needed during the extraction from the source systems 110 and loading of the datamart 150, and (3) the information used in the querying of the datamart 150 and supplying the result sets. The relationships between the elements of the metadata 160 are described in greater detail below. However, the following provides brief descriptions of these elements.

Detailed Description Text (95):
The schema definitions 161 hold the definition of the schema for the datamart 150. Typically, a consultant, using the consultant computer 190, can interface with the enterprise manager 102 to define the schema definition 161 for the datamart 150. In particular, the consultant can use the enterprise manager interface 192 to define a star schema for the datamart 150. This star schema is organized around the business processes of the business for which the datamart is being created. What is important is that the consultant can easily define a schema for the datamart 150 and that definition is kept in the schema definitions 161. From the schema definitions 161, not only can the tables in the datamart 150 be generated, but also the automatic extraction and conversion of the data from the source systems 110 can be performed, aggregates are set up, and a query mechanism is generated.

Detailed Description Text (96):
The connectors 162, the semantic definitions 163, and the source system information 164, are all related to the extraction of the data from the source systems 110. The connectors 162 define the access routines for extracting the source systems data 110. The semantic definitions 163 define how that extracted data should be converted when it is loaded into the datamart 150. The semantic definitions 163 provide important advantages to the system 100. In particular, the semantic definitions 163 allow for a simplified definition of the datamart 150, consistent meaning of the data in the datamart 150, and allow for complex changes to the schema to be easily propagated to the datamart 150. The source system information 164 defines how to extract the data from the systems 110.

Detailed Description Text (97):
The aggregate information 167 defines how data in the datamart 150 is treated once it is extracted. The aggregate information 167 allows for the creation of aggregates. Aggregates are aggregations of various fields of data in the datamart 150. Aggregates support more complex and powerful queries to be executed on the datamart 150. The aggregates also improve the performance of the system during the querying process and allow for time navigation of the data in the datamart 150. Time navigation is the process of creating backlog result sets by hopping through date aggregates from the beginning of time in the datamart 150 to the present.

Detailed Description Text (98):
The measurement information 168 and the query/reporting information 169 support the querying of the datamart 150. A measure is a piece of numeric data in the datamart 150 that is useful to a user. That is, individual fact columns from source systems can be very implementation specific. These columns may not correspond to what users would prefer to see. For example, a user may want to see a net price added with a total cost. However, the fact table may only include the net price or the total cost. The measure information 168 allows the consultant to define the abstract notion of the calculation associated with the net price added to the total cost.

Detailed Description Text (99):
In some embodiments of the invention, the metadata 160 also includes security information. The security information defines the level of access for various users to the various tables and fields in the datamart 150. This security information automatically restricts access to that data.

Detailed Description Text (101):
The system 100 can be implemented on a network of computers running Windows NT and UNIX. The datamart 150 can be implemented on top of an Oracle (SQL Server, or ODBC)

database. However, this physical structure of the system 100 can be implemented in any number of ways, and the invention does not require this specific hardware configuration.

Detailed Description Text (102):
The enterprise manager 102 is a program that is responsible for supporting the definition of the schema, and the creation of the tables in the datamart 150 from the schema definitions 161. The enterprise manager 102 also controls the extraction program 120. (In some embodiments, the extraction program 120 and the semantic template conversion program 140 are included in the enterprise manager 102). During the execution of the extraction program 120, the extraction program 120, the staging tables 130, the semantic template conversion 140, and the datamart 150 are all used. The extraction program 120 uses the connectors 162 and the source system information 164 to extract the information from the source systems 110. The extracted data is loaded into the staging tables 130.

Detailed Description Text (103):
The staging tables 130 are temporary tables used to hold the source system data before performing any semantic conversions on that data. The staging tables 130 also allow for the conversion of the source system data prior to moving the data into the datamart 150.

Detailed Description Text (104):
Once the staging tables 130 have been loaded, the semantic definitions 163 can be accessed from the enterprise manager 102 to convert the information in the staging tables 130 to predefined data semantics. These predefined data semantics allow for powerful queries, consistency in the definition of the meaning of the data in the datamart 150, and allow for changes to be made to the schema. Generally, the semantic template conversion 140 takes data stored in the staging tables 130, performs a conversion of that data according to a corresponding semantic definition (defined in the schema definitions 161), and populates the datamart 150 with the converted data.

Detailed Description Text (105):
Importantly, the predefined data semantics substantially simplify the creation and population of the datamart 150. In previous systems, the consultant would have to implement all of the data manipulation and population programs by hand. By selecting a particular semantic definition for a particular fact, or dimension, in the schema, the consultant has automatically defined the access and manipulation for populating programs for that table. Allowing the consultant to select a predefined data semantic not only reduces the tedious coding previously required of the consultant, but also allows for the automatic insertion of foreign keys, transaction types, date, and other information into the schema, and therefore the datamart 150. This additional information causes the datamart 150 to be well-formed.

Detailed Description Text (106):
The aggregate builder 170, as mentioned above, aggregates data in the datamart 150 according to the aggregate information 167 and the schema definitions 161. The results of the aggregate builder 170 allow for more powerful and faster queries to be performed on the datamart 150.

Detailed Description Text (107):
The query/reporting program 104 supports the querying of the datamart 150 and presents results of those queries. The query and reporting process 104 uses the measurement information 168 and the query and reporting information 169, in addition to the schema definitions 161, to query the datamart 150 and provide that information to the web server 186. The query/reporting. information 169 includes filters and form definitions. The filters allow the user to filter different fields out of the datamart 150. The forms allow the users to indicate which fields a user

is particularly interested in.

Detailed Description Text (108):
The metadata 160, although including many different types of definitional data, importantly includes the schema definition 161 and the semantic definitions 163. The enterprise manager 102 can use the schema definitions 161 to build the tables in the datamart 150. Through the combination of these two pieces of metadata 160, the enterprise manager 102 can take data from a source system 110, perform semantic conversions on that data and populate the datamart 150. Thus, in some embodiments of the invention, the system includes only the schema definitions 161 and the semantic definitions 163.

Detailed Description Text (110):
The source systems 110, as defined above, represent large databases from which data for the datamart 150 is pulled. Examples of such systems include large on line transaction processing (OLTP) systems. Typically these source systems 110 are relational databases having multiple tables and relations between those tables. The source systems 110 do not generally support powerful queries that provide high level information about the business in which the source systems 110 are used. Thus, the system 100 is used to extract the data from the source systems 110 and to provide an improved schema for querying that data. In some embodiments, the source systems 110 include non-relational databases such as object databases. In other embodiments, the source systems 110 can include flat file systems or combined relational and object databases. What is important is the source systems 110 can provide data to the system 100 through the connectors 162 and the source system information 164.

Detailed Description Text (112):
The user can access the web server 186 through the user computer 180. In this example, the user computer 180 can access the web server 186 through an HTTP connection. The user computer 180 sends a file request to the web server 186. This file request represents a request for a query of the datamart 150. The web server 186 runs a Java program upon receiving the request. The query and reporting program 104 converts the information from the Java program into a query that the datamart 150 will understand. In one embodiment, the query/reporting program 104 converts the query into a set of SQL statements. The SQL statements are run against the datamart 150. The results of the statements are processed and provided back to the user computer 180.

Detailed Description Text (113):
Example Method of Defining and Using the Datamart

Detailed Description Text (114):
FIG. 2 illustrates an embodiment of a method of defining the datamart 150, loading the datamart 150, and then accessing the data in the datamart 150. This example can be broken into four subparts: a build datamart process 202, an extraction and loading process 204, a build aggregates process 205, and a query and reporting process 206. This example can be implemented using the system 100.

Detailed Description Text (115):
At block 210, a consultant uses the enterprise manager 102 to define the schema. The schema is defined using the metadata 160. This process is illustrated in greater detail in FIG. 7 through FIG. 35. Generally, defining the schema involves determining the business processes of the organization for which the system 100 is being implemented. The consultant then defines the star schema for those business processes. The star schema has a fact table and a number of dimensions. The consultant also defines from where the data in the schema is to be derived. That is, the consultant defines from which fields and tables the information is to be extracted from the source systems 110. The consultant also defines how that data is to be put into the datamart 150. That is, the consultant associates each piece of

data with a semantic meaning. This semantic meaning defines how the data from the source system is to be manipulated and how it is to populate the datamart 150. At this point, the consultant can also define the aggregates that can be used in the datamart 150.

Detailed Description Text (116):
Once the datamart 150 has been defined, it can then be automatically built. At block 220, the enterprise manager 102 generates table creation SQL statements according to the definition of the metadata. In one embodiment of the invention, block 220 is accomplished by performing queries on the schema definitions 161 to generate the fact table creation statements, the fact staging table creation statements, the dimension table creation statements, the dimension staging table creation statements, and the dimension mapping table creation statements. These tables are described in greater detail below. From the results of these queries, SQL CREATE TABLE statements are created. Importantly, the schema definitions 161 provide the information the enterprise manager 102 needs to build the datamart 150.

Detailed Description Text (117):
Note that this process can also be used to modify the schema of an existing datamart 150. Therefore, at block 220, the SQL tables being created will cause the existing datamart 150 to be modified without losing the data in the datamart 150.

Detailed Description Text (118):
At block 230, the enterprise manager 102 issues the table generation statements to the database upon which the datamart 150 is being created. That database creates the tables, which correspond to the datamart 150. After block 230, the build the datamart process 202 is complete.

Detailed Description Text (119):
Now the extraction process 204 can be performed. The extraction process 204 is run on a periodic basis to load data from the source systems 110 into the datamart 150. This process can be run multiple times for the datamart 150.

Detailed Description Text (120):
At block 260, the connectors 162 are used by the enterprise manager 102, and in particular, they are used by the extraction program 120 to extract the data from the source systems 110. The connectors 162 can include SQL statement templates (not to be confused with semantic templates, as described below) for extracting data from the source systems 110. The extraction program 120 uses these templates, in addition to the source system information 164, to generate SQL statements. These SQL statements are issued to the source system 110 and the results are loaded into the staging tables 130. (The staging tables 130 had been created as a result of block 230.) Once the staging tables have been loaded, the data can then be moved into the datamart 150.

Detailed Description Text (121):
At block 270, the staging table data is moved into the datamart 150 using the semantic definitions 163. The semantic definitions 163 are templates for converting the staging tables 130 data according to predefined data semantics. These predefined data semantics, as described below, provide semantic meaning to the data being loaded from the staging tables 130. Note that the data from the staging tables 130, as processed by the semantic template conversion 140, is placed in the tables in the datamart 150.

Detailed Description Text (122):
Thus, the schema definition and the semantic definitions 163 are used to generate and populate the datamart 150 such that the datamart 150 is well-formed. Examples of the well-formedness of the datamart 150 are as follows. (1) Two columns related by a relational join will be from the same domain. (2) If table A has a many-to-one

relationship to table B, then table A has a foreign key that corresponds to table B. (3) A many-to-many relationship, between two tables A and B, is always expressed by an associative table that is created in a uniform way. For each unique many-to-many relationship, a unique value is created in the associative table and reused whenever that many-to-many relationship occurs. Denormalization is always done correctly. (4) Pulling information from one table to be put into another table, for access efficiency, is done correctly. Previous systems cannot guarantee such a well-formed database system because hand coding of the creation and population operations is required. This hand coding can easily introduce errors into datamart creation and population processes.

Detailed Description Text (123):
Once the extraction process 204 has completed, the aggregates can be built in the build aggregates process 205. The aggregates are tables of pre-calculated combinations of dimensions and facts. Importantly, they greatly increase the speed of queries. Generally, the aggregate definitions, stored in the aggregate information 167, are accessed and built using the aggregate definitions (which interface with the schema definitions). At block 275, the aggregate builder 170 accesses the metadata 160 to build the aggregates. Often, the aggregate building is done at night.

Detailed Description Text (124):
After aggregates are built, the querying and reporting process 206 can be performed. The querying and reporting process 206 can be performed anytime after the creation of the datamart 150. Importantly, when an aggregate is created, the appropriate operation for that aggregate is used. For example, revenue elements are added to produce an aggregate, while daily account balances are averaged to produce an aggregate.

Detailed Description Text (125):
At block 277, the consultant defines the query mechanism schema for the system 100. In particular, the consultant defines the query/reporting information 169 and the measurement information 168. These two pieces of metadata 160 allow the system 100 to report meaningfully consistent information to users. Also, the consultant is not burdened with having to hand create the possible reports.

Detailed Description Text (126):
At block 280, a query is generated. In one embodiment of the invention, the query is generated at the query/reporting program 104. In other embodiments, the query can be generated at the user computer 180 through the HTTP, web server 186, Java coupling to the query/reporting program 104. What is important here is that some query is generated that can be used to access the datamart 150. Importantly because the schema definitions 161 are available to the query and reporting program 104, the user can be presented with forms from which a query can be easily and automatically generated.

Detailed Description Text (127):
At block 290, the answer set (the results) is created by the datamart 150. This answer set is then propagated back through the query/reporting program 104, and ultimately to the user computer 180. The results are formatted according to the query/reporting information 169.

Detailed Description Text (128):
Top Level Metadata Schema

Detailed Description Text (129):
As noted in the background, multi-dimensional datamarts use star schemas. The system 100 uses star schemas in a larger organization that allows for the sharing of dimension tables by sets of similar facts. This larger organization is called a constellation. FIG. 3 illustrates a schema for the schema definitions tables that

support constellations. (The schema of FIG. 3 is labeled the schema for schema definitions 300.) That is, FIG. 3 illustrates a schema used in the system 100 to define schemas for the datamart. FIG. 3 also illustrates some of the aggregate information 167 schema.

Detailed Description Text (131):
It is important to remember that FIG. 3 through FIG. 5 illustrate the schema of the system used to generate and run the datamart 150. Rows in these tables define the schema for use in the datamart 150. From these rows, create table, table query, etc., commands are created. These commands are used to create the tables in the datamart 150 and to access that datamart.

Detailed Description Text (132):
Also, as mentioned previously, the datamart 150 is well-formed because, among other reasons, the system 100 automatically includes additional columns in the table created in the datamart 150. For example, source system key, foreign key, and time and date columns are automatically added (where appropriate). The rest of the elements of the system can then rely on the existence of these columns. This prevents, for example, the creation of an inconsistent schema where only some of the tables include date and time information.

Detailed Description Text (134):
Top Level Metadata List

Detailed Description Text (136):
Top Level Metadata Descriptions

Detailed Description Text (137):
It is important to remember that the tables in FIG. 3 are only used to define the schema in the datamart 150. Thus, a fact table 304 in FIG. 3 is not the actual fact table in the datamart 150, but the definition of that fact table. Each row in a table corresponds to an instance of that table.

Detailed Description Text (138):
The constellation 302 defines the organization of the schema in the datamart 150. It is the top level table in the schema definition.

Detailed Description Text (140):
The fact table 304 defines the metadata 160 table describing all of the fact tables within a given constellation 302. The attributes of the fact table 304 include a build aggregates flag, a cleanse flag, a constellation key, a description, a fact table key, a fact table name, and a truncate stage flag. Each attribute corresponds to a column in the fact table 304. The build aggregate flag indicates whether or not to build aggregates for a particular fact on the next execution of the aggregate builder 170. The cleanse flag is a flag that is used in many of the tables to obliterate the actual measures within a table in the datamart 150 (particularly useful in demonstrations of the system 100 where sensitive data would otherwise be revealed). The constellation key points to the parent constellation 302 for a given fact table 304. The fact table name is the name of the fact table used in constructing the corresponding physical table names in the datamart 150. The truncate stage flag is used to indicate whether or not to truncate the fact staging table on the next extraction.

Detailed Description Text (141):
The fact column 310 lists all of the fact attributes within a single fact table 304. The fact column 310 includes a cleanse flag, a description, a fact aggregate operator, a fact column key, a fact column name, a fact column number, a fact table key, and a physical type. The fact aggregate operator is an SQL operator used to aggregate this fact column in the datamart 150. The fact column key is the primary key for the fact column. The fact column name is the physical name of the fact

column. The fact column number counts and orders the number of columns in the fact table. The fact table key points to the fact table to which the corresponding fact column belongs. The fact table key points to the fact table to which the fact column belongs. The physical type is the database type for the fact column. This type is a logical type and provides for independence of implementation of the datamart 150 from the underlying database used.

Detailed Description Text (145):
The dimension base 306 is the metadata 160 describing all the dimension tables that can be used in a given constellation 302. These dimension bases can then be used in multiple constellations. The dimension base 306 includes the following attributes: an aggregate key operator, a cleanse flag, a description, a dimension base key, dimension base name, a dimension base type, and a truncate stage flag. The aggregate key operator is an SQL operator used by the aggregate builder 170 to build aggregates from a dimension. The cleanse flag and description act similarly to those attributes in other tables. The dimension base key is the primary key for the dimension base 306. The dimension base name is the name of the base dimension used in constructing real tables in the datamart 150. The dimension base type indicates the type of a dimension base (either default or special (special includes "date" and "transaction type," which are used by the system 100). The truncate stage flag operates in the manner similar to other truncate stage flags.

Detailed Description Text (146):
The dimension column 329 defines the list of dimension attributes that are valid for a single base dimension 306 and inherited by a dimension usage. The dimension column 329 includes a cleanse label, a cleanse map key, a cleanse type, a description, a dimension base key, a dimension column key, a dimension column name, a dimension column number, a dimension number key, grouped by field, a physical type, a primary key, a time navigation field, and a default value. The cleanse label is a label presented to users after this column has been cleansed. The cleanse map key is for use when cleansing using value mapping. The cleanse map key indicates the mapping group to use. The cleanse type is the method for cleansing the dimension column 329. The description is for documenting the dimension column 329. The dimension base key is the numbered base in which the column resides. The dimension column key is the primary key for the dimension column 329. The dimension column name is the physical name of the column. The dimension column number is the count of the dimension columns (to prevent too many from being created in the datamart 150). The dimension node key is the aggregate hierarchy group in which the column resides. The "group by" field is used for special dimensions to indicate whether or not this column needs to be "grouped by" during the processing by the aggregate builder 170. The physical type is a logical database type for this dimension column 329. The primary key is used in special dimensions to indicate whether or not this column is the primary key. The time navigation field is for the date special dimension to indicate whether or not time navigation should use this field. The default value is the default value for the dimension column.

Detailed Description Text (148):
The dimension role 320 is a metadata 160 table that describes all of the dimension tables used in a constellation 302. The dimension role 320 includes a constellation key, a degenerative number, a description, a dimension base key, a dimension role key, a dimension role name, and a dimension role number. The constellation key points to the constellation 302 in which the dimension role 320 resides. The degenerative number defines the order of degenerate columns within fact tables in a constellation. The description is a documentation field for describing a dimension role. The dimension base key is the dimension base that this dimension role refers to. The dimension role key is the primary key for the dimension role 320. The dimension role name is the name of the dimension role and is used when constructing the foreign keys in the fact tables in the datamart 150. The dimension role number defines the order of the dimension roles within a constellation. That is, a constellation may have multiple dimension roles and the dimension role number

allows for an ordering of those dimension roles.

Detailed Description Text (153):
The semantic instance 308 is a single record that represents the manner in which a fact or dimension table is extracted from staging tables, manipulated, and then used to populate the corresponding table in the datamart 150. The semantic instance 308 includes an extraction node key, dimension base key, a fact table key, a semantic instance key, and a semantic type key. The extraction node key points to the extraction node that a particular semantic instance belongs to. The dimension base key is the dimension base table owning this semantic instance. The fact table key points to the fact table owning this semantic instance. Only one of the dimension base key and the fact table key is filled in for a semantic instance 308 because the semantic instance can only be applied to one or the other. The semantic instance key is a primary key for the semantic instance 308. The semantic type key is the indicator of the type of transformation necessary to construct this type of semantic instance in the datamart 150.

Detailed Description Text (156):
The aggregate group 342 defines a set of aggregates to be built for a constellation. An aggregate group 342 will cause a combinatorial creation of many aggregate tables in the datamart 150. The consultant defines for which dimensions aggregates are to be built (e.g., the consultant will define that one, none, all, etc. columns of a dimension are to be aggregated on in an aggregate group). The aggregate filtering done by the query and reporting program 104 will select the most appropriate aggregates for a given query.

Detailed Description Text (162):
A special dimension base 391 provides details about special built-in dimensions in the system 100. The special dimension base includes an "always include an aggregate" field, a default aggregate dimension type, a dimension base key, a list order in fact, a physical type of key, an index flag, and a special dimension base key. The "always include an aggregate" field indicates whether or not this dimension table must always be included in all aggregates. The default aggregate dimension type is the default manner in which this dimension is included in aggregate groups. The dimension base key is the one to one relationship to a dimension base. The list order in fact is the order in fact tables that the foreign key to this table will be listed. The physical type of key is the logical database type that foreign keys in the fact tables that point to this special dimension will be. The index flag is used in indexing. The special dimension base key is the primary key for the special dimension base 391.

Detailed Description Text (164):
The physical type 330 defines a look up table of logical data types that are relational database management system (RDBMS) independent. The physical type 330 is a logical data type that works across various source systems storage types. The physical type 330 includes a database physical type, a default value, and a special type.

Detailed Description Text (171):
The metacolumn 334 is a column that occurs by default in tables in the datamart 150. The metacolumn 334 includes an actual table type, a list order, a metacolumn key, a metacolumn name, and a physical type. The actual table type indicates the type of physical table in which this special column should appear. The list order is the order this column occurs in tables of the appropriate type. The metacolumn key is the primary key. The metacolumn name is the physical name of the column when it is used. The physical type is the logical data type for this column.

Detailed Description Text (172):
The actual table type 336 is a look up table for actual table types. Actual table types can be fact, dimension stage, fact stage, dimension map, or dimension.

Detailed Description Text (173):
The aggregate group 342, the aggregate fact 340, the aggregate dimension set 372, the dimension column set 370, the dimension column set definition 374, the fact index 380, the fact index definition 384, and the fact index number 382 are for future use and are therefore optional. Each of these tables provides greater flexibility when defining the metadata 160, improves the performance of the system 100, or may otherwise enhances the system 100.

Detailed Description Text (174):
Top Level Metadata Use

Detailed Description Text (176):
It is important to note that many of the tables in FIG. 3 are actually used in providing layers of abstraction to allow for the reuse of information and non-abstract tables. Therefore, a consultant will often only deal with only some of the tables in the FIG. 3. For the purposes of describing how the metadata 160 can be used to define a schema for the datamart 150, these grouping and levels of abstraction tables will be described where appropriate.

Detailed Description Text (177):
Generally, a consultant will create a new datamart 150 by defining instances of the dimension bases 306, and constellations 302. Each instance corresponds to a row in the dimensions bases 306 table or the constellation 302 table. The constellation instances are defined by defining aggregates, dimensions, facts, measures, and ticksheets. The following describes the definition of a schema using the metadata 160. This corresponds to block 210 of FIG. 2.

Detailed Description Text (178):
Beginning with the facts in a constellation, the consultant defines a fact table 304 row that will define the hub table in a star schema supported by the constellation. Again, it is important to remember that the fact tables in FIG. 3 are for definitional purposes, and are not the real fact tables in the datamart 150. A row in the fact column 310 holds the details of what columns will be created for place holders of actual values in a corresponding fact table. Thus, for each fact, the consultant defines the various fact columns.

Detailed Description Text (180):
Remember that the dimension base 306 holds the information to define the actual dimensions of the tables in the datamart 150. The dimension role 320 allows for the reuse of the dimension base tables. Thus, different dimension roles can refer to the same dimension base. This provides an important feature of some embodiments of the invention where the same dimension bases can be used in multiple constellations or within the same constellation. The dimension columns 329 define the columns on which queries can be performed in the datamart 150. The dimension node table 326 helps relate the dimension columns 329. Thus, the consultant will have defined the basic schema for the datamart 150.

Detailed Description Text (181):
The aggregate group 342 defines how particular facts or dimensions are to be aggregated by the aggregate builder 170. These aggregated facts provide much faster queries in the datamart 150.

Detailed Description Text (182):
The cleansing map tables are for scrambling the data in the datamart 150 for presentations to people who want to see the functionality of the system 100, without having to reveal the actual data in the datamart 150.

Detailed Description Text (183):
The special dimensions are the transaction type table and date values that are

included in every fact table. Because this is included in every fact table, the system 100 can rely on the existence of the transaction type during the various stages of datamart 150 creation, modification, querying, and the like.

Detailed Description Text (184):
Thus, the elements of FIG. 3 can be used to allow the consultant to define the schema definitions 161 for creating the tables in the datamart 150.

Detailed Description Text (185):
Extraction Metadata

Detailed Description Text (186):
The following describes the metadata 160 used in the extraction process 204. This metadata, represented as extraction schema 400, is shown in FIG. 4. The extraction process focuses around the job and connector tables. In general, these tables define the various steps in extracting the source system data into the staging tables 130 and performing the desired semantic conversions on that data.

Detailed Description Text (187):
Extraction Metadata List

Detailed Description Text (189):
Extraction Metadata Descriptions

Detailed Description Text (191):
The job 402 is a top level object for controlling the work flow during the extraction and loading process 204. The job 402 includes a check databases field, a check tables field, a description, a label, an initial load flag, a job key, a job name, a log file width, a mail to on error, a mail to on success, and a truncate flag. The check databases field indicates whether or not an attempt should be made to log into all the data stores before executing the job. The check tables flag indicates whether or not to check for the existence of all the tables in the datamart 150 before executing the job. The description is for documenting the job (usually done by the consultant). The enabled flag indicates whether or not a particular job can be run. The initial load flag indicates whether or not to ignore all previous time stamped constraints when running a particular job. The job key is the primary key for the job table 402. The job name is the internal name of the job. The log file width indicates how many characters wide to make rows in the log file output. The mail to on error, and the mail to on success indicate where E-mail messages should be sent after failure or success of the particular job. The truncate flag indicates whether or not to truncate any tables when running a job.

Detailed Description Text (198):
The connector time stamp 407 relates to information about incremental extraction. An incremental extraction is where increments of the data in the source system 110 are extracted. The connector time stamp includes a connector key, a connector time stamp key, current max date, a current max time stamp, a last max date, and a last max time stamp. The connector key points to the connector to which the connector time stamp applies. The connector time stamp key is a primary key. The current max date is an indicator of the proposed new system date of the last successful extraction. The current maximum time stamp is the proposed new SQL server time stamp field for the last successful extraction. The last maximum date is the system date of the last successful extraction. The last maximum time stamp is the SQL server time stamp field for the source system databases at the last successful extraction.

Detailed Description Text (200):
The connector column latch 409 defines information about incremental extraction based on a database column. The incremental extraction information is thus kept in the database and can be retrieved. The connector column latch 409 includes the

following attributes: a column name, a connector column latch key, a connector key, a current maximum value, a last maximum value, and a table name. The table name is the name in the input data store for the corresponding connector. The column name is the column name within that table. The connector column latch key is the primary key. The connector key points to the connector to which this latch applies. The current max value represents the proposed new maximum value for the incremental extraction. This number is pushed into the last maximum value if the currently executing extraction succeeds. The last maximum value is the maximum value that was extracted during the last run of the extraction.

Detailed Description Text (204):
The extraction node 410 is a single node, or step, in the extraction tree. The extraction tree defines the order of extraction steps. An extraction node includes an extraction node type, a debug level, a debug level row, an enabled flag, an extraction group key, an extraction node key, a list order, and on error type, a parent extraction node key, and a phase. The extraction node type defines the type of extraction node (as defined in the extraction node type table 491). This relationship is important and allows for the conversion of data in the staging tables for use in the datamart 150. The debug level indicates how to debug a particular step during execution. The debug level row indicates which row to start debugging at for SQL statements. The enabled flag indicates whether or not to execute a particular SQL statement associated with the extraction node. The extraction group key points to, if not null, the name of the group. The extraction node key is a primary key. The list order and the phase define the order of the corresponding step within an extraction node's parent. The on error type indicates what to do if there is an error during the execution of the step associated with the extraction node. The parent extraction node key points to the parent extraction node of the present extraction node.

Detailed Description Text (210):
The external column 424 defines a column in a user defined extraction table. The external column 424 includes the attributes for documenting a particular external column, and the column name in the external table. A pointer to the external table, a list order of appearance in the external table, and a physical type of the logical database for this column are included in the external column 424.

Detailed Description Text (215):
The semantic type 430 defines a set of predetermined semantic types for use in defining a schema. The semantic type includes a logical name for a particular transformation. Associated with the semantic type are a dimension semantic type 432 and a fact semantic type 434. The dimension semantic type table 432 defines the ways in which dimension data in the staging tables 130 can be extracted and put into the datamart 150. Similarly, the fact semantic type defines the ways in which the information in the staging tables 130 can be put into the fact tables of the datamart 150. Both the fact semantic type 434 and the dimension semantic type 432 include pointers to an actual table type and are used to subset the full list of semantic types.

Detailed Description Text (217):
The adaptive template 438 is a semantic transformation template (e.g., an SQL program) that is used in the extraction of the data in the staging tables 130 to turn all source data into transactional data. The adaptive template 438 includes attributes indicating a logical name for an adaptive program used within semantic transformations.

Detailed Description Text (221):
The data store 440 defines a logical data source, or sink, used during the extraction. The data store 440 includes the following attributes: the data store key, a data store flag, a description, a name, a source system key, and a store type. The data store key is the primary key. The datamart flag indicates whether or

not this data store is the special datamart store. Since the datamart 150 and the metadata 160 can reside in the same database or different, the data mark helps resolve the location of the datamart 150. The description is for documentation of the particular data store. The name is the logical name of the data store. The source system key points to the source system identifier to which this data store belongs. This allows live, and backup, source systems to share the same identifier. The store type indicates the store type of this data store.

Detailed Description Text (222):
The source system 442 is a logical identifier for a source system 110 from which data can be pulled. This allows two physical databases to act as one master database and one backup, for example. The source system 442 includes a description attribute, a source system key and a source system name. The description is for documentation to describe the source system. The source system key is a primary key for this table. This number also becomes identified source system field in the staging tables 130 being filled. The source system names is a logical name for a source system 110 from which the system 100 is pulling data.

Detailed Description Text (225):
The Oracle store 454 defines information about particular Oracle databases. The Oracle store 454 includes the following attributes: a data store key, an instance name, an Oracle store key, a password, an SQL network name, a user name, and a version. The data store key is a one to one relationship key to the data store being defined. The Oracle store key is the primary key. The password and user name are used to access a specific Oracle system. The version number is the Oracle vendor version number. The SQL network name is the SQL net instance name. The store version is a version of the store type (the database vendor's version for example) that the system recognizes. The store version has a pointer to the store type being defined and also includes a version number attribute.

Detailed Description Text (226):
The SQL server store table 456 defines details about an SQL server system. The SQL server store includes the following attributes: a data store key, a database name, a password, a server, and SQL server store key, a user name, and a version. The data store key is a one to one relationship to a data store entry. The database name is an SQL server database name ($$DEFAULT means the database in which this role resides). The password is the SQL server password. $$ DEFAULT again means the password currently logged into to read this data. The server is the SQL server name. The SQL server store key is the primary key. The user name is the SQL server user name. The version is the vendor's version number of this SQL server. $$DEFAULT means use the default value for the current database being used. For example, the database name means the database in which this role resides.

Detailed Description Text (227):
Extraction Metadata Use

Detailed Description Text (228):
The following describes the tables of FIG. 4 in the context of the extraction process 204. The job, the job step, and the connector, group extraction steps for extracting information from the source systems 110 and cause that information to be placed in the datamart 150. This organization allows for a very flexible extraction process. For example, where a two phase extraction is required, one connector could be used to extract the information from the source system 110, while a second connector could then be used to take this extracted data from an external table.

Detailed Description Text (230):
The following is an example illustrating the organization of job. Assume that a consultant wants to extract information from a source system that provided a raw set of home addresses. A system call could be run as part of a job step. The system call would determine the zip codes associated with those addresses. The zip codes

could then be included in the datamart 150.

Detailed Description Text (233):
An SQL statement is a single step in an extraction run that represents a data push or a data pull. The SQL source code dictates the action for a given extraction node. After the SQL statements are run, the staging tables 130 are ready. The semantic conversion of the data in the staging tables 130 can occur.

Detailed Description Text (234):
The semantic instance represents the use of a single generic template on one fact or dimension table. The semantic type associated with the semantic instance is one of a number of pre-defined recognized data meanings within the system 100 (e.g., an "order"). The semantic types correspond to programs for converting the data in the staging tables 130 into data for use in the datamart 150. An example of a semantic type is a "slowly changing dimension" type.

Detailed Description Text (235):
The semantics types, as mention previously, are made up of a series of templates. These templates include tokens that can be replaced with information from the corresponding dimension base or fact table. An example of an adaptive template is one that would be used in re-indexing of a fact table. This could be used as the last step in the semantic transformation of facts. The re-indexing will help speed the operation of the datamart 150. Importantly, this same indexing is performed for each fact table. No matter which semantic type is chosen for a given fact table, the same indexing is performed. Thus, this adaptive template can be used in each semantic type through its semantic type definition.

Detailed Description Text (242):
In some embodiments, the pre-parsed templates include additional tokens to deal with specific data stores. For example, the "select into" statement is a token in the pre-parsed version. This compensates for whether the data store is in Oracle database or an SQL server.

Detailed Description Text (246):
As mentioned previously, the use of the semantic types significantly reduces the amount of work needed to implement the datamart 150. By selecting a semantic type for a particular fact table or dimension table, the consultant automatically selects the corresponding pre-parsed SQL adaptive templates. The selected adaptive templates are then automatically converted into post-parsed SQL statements that include the schema specific information for the datamart 150. Additionally, these post-parsed SQL statements include the SQL for converting the data in the staging tables 130 into data that can be used in the datamart 150 tables.

Detailed Description Text (249):
The team template is used to properly populate an "associative" dimension table. Such a table is used whenever there is a one-to-many relationship between an individual fact row and a dimension. For example, if multiple salespeople can split the credit for an order, one needs some way to represent this situation in the datamart. In a star schema, one normally associates a tuple of dimension values with a fact row (e.g. product, customer, salesrep dimensions for the fact row containing price, quantity etc.). Since there is only a single salesrep.sub.-- key, one could normally have only one salesrep associated with this transaction. There are two solutions. One is to introduce multiple fact rows for a transaction invovling one to many relationships. If there were three salesreps on a specific order, there would be three fact rows for this order stored in the database. This has the disadvantage of multiplying the data size by a factor of three and slows queries. Also queries that are concerned with the total number of transactions become more difficult to process since duplicate rows, due to the multiplication by the number of salesreps, must be eliminated.

Detailed Description Text (250):
Another solution is to introduce an associative table between the actual salesrep dimension table and the fact table. Conceptually, the associative table contains "teams" of salespeople. If salesreps A, B and C often sell products together, they will be associated with a unique team key. The team key will be stored in each fact row for orders sold by the A, B, C team. The associative table will associate the team key with the three rows for A, B and C in the salesrep table. The associative table will have 3 rows representing this team (A-key, team1-key), (B-key, team1-key) and (C-key, team1-key). If the team of A, B, D and Q also sold products together, the associative table would have four additional rows (A-key, team2-key), (B-key, team2-key), (D-key, team2-key), (Q-key, team2-key). The team template scans the staging table used to load the fact table and generates the appropriate rows for entry into the associative table, only for those teams THAT ACTUALLY OCCUR in the fact rows being loaded.

Detailed Description Text (258):
The population of both the denormalized team dimension and the associative table are difficult to code properly. This is especially true if this is done incrementally (e.g., on nightly extracts) and if you want to be independent of team order (e.g. A, B, C) is the same as (A, C, B). Thus, allowing the consultant to simply select this data semantic provides a significant improvement over previous systems.

Detailed Description Text (259):
Runtime Metadata

Detailed Description Text (260):
FIG. 5 illustrates the schema for the runtime environment within the system 100. The runtime schema 500 represents the schema description for the schema of the running datamart 150. That is, when the datamart 150 is created or modified, the schema definition is propagated into the runtime schema 500. Thus, the runtime schema 500 allows for the datamart 150 to be changed without having to rebuild all the tables and repopulate all of those tables. Additionally, the runtime metadata 500 provides the support for aggregate navigation. Aggregate navigation involves determining which aggregate to use in response to a query. Schema modification and aggregate navigation will now be explained in greater detail.

Detailed Description Text (261):
The schema modification involves comparing the changed schema definition with the present schema definition. As will be seen below, an actual table 502 keeps track of all of the dimension tables and the fact tables in the datamart 150. When a change is made to the schema definition, a comparison is made between the old definition and the new definition. The difference between these definitions defines the set of tables, columns, and rows that need to be added, deleted or modified, in some way. Importantly, the modifications can often be made without losing any data in the datamart 150.

Detailed Description Text (262):
The aggregate navigation process determines which aggregate most closely suits a particular query. The runtime metadata 160 keeps track of the aggregates available in the datamart 150. The query and reporting program 104 initiates a view of the runtime metadata 500 (in particular, the tables holding the aggregate tables definitions). The view results indicate which aggregates are available to answer the particular query. The view results are further examined to determine the best aggregate to use (the one that most closely corresponds to the query).

Detailed Description Text (265):
Runtime Metadata List

Detailed Description Text (266):

The runtime schema 500 includes the following elements: an actual table 502, an actual column 504, a fact aggregate table 512, a fact aggregate dimension 514, a dimension base aggregate 516, a dimension base aggregate column 518, a datamart letter 510, the dimension base 306, the fact table 304, the external table 422, an actual column 504, a physical type definition 530, an actual table type 336, an actual column type 540, the physical type 330, a database physical type 595, the translation string 332, a translation actual 539, a store type 450, a date 560, a business process 570, an adaptive template profile 580, and a transaction type 590.

Detailed Description Text (267):
Runtime Metadata Descriptions

Detailed Description Text (268):
The actual table 502 corresponds to metadata 160 that describe which dimension base and fact tables "actually" exist in the datamart 150.

Detailed Description Text (269):
The actual table 502 includes the following attributes: an actual table key, an actual table name, an actual table type, a dimension base key, an external table key, a fact table key, an index flag, a mirror flag, and a logical table name. The primary key is the actual table key. The actual table name corresponds to the physical name of this table in the database implementing the datamart 150. The actual table type is the logical type of this physical table. For example, if this is a dimension staging table or a fact staging table. The dimension base key points to the dimension base table definition that defined the corresponding physical table. The external table key points to the external table definition that defined the physical table. The fact table key points to the fact table definition that defined the corresponding physical table. The index flag and the mirror flag are used in indexing and mirroring, respectively. The logical table name defines the logical name for this table.

Detailed Description Text (270):
The actual column 504 is metadata describing a physical column in a physical table in the datamart 150. The actual column table latches this definition information when the physical tables are built in the datamart 150. The actual column 504 includes the following attributes: the actual column key, an actual column name, an actual column type, an actual table key, a dimension role name, a foreign table key, a group by field, a hierarchy, a list order, a parent hierarchy, a physical type, a primary key, and a time navigation field. The actual column name is the name of the physical column in the physical table in the datamart 150. The actual column type is the logical type of the column. The actual table key points to the actual table in which the actual column lives. The dimension role name is the logical role name of the dimension in the fact table for dimension foreign keys inside of a fact table. The foreign table key points to the actual dimension base tables in the actual tables 502 (the foreign table key is applicable to fact actual columns that are foreign keys to dimensions). The group by field, for dimension table, is true when this column should be included in an aggregate builder group. The hierarchy for dimension, for dimension columns, indicates that aggregate builder group to which this column belongs. The list order is the order of the column in the actual table 502. The parent hierarchy, for dimension columns, indicates the parent aggregate builder group to which this column belongs. The physical type is a logical data type of the column. The primary key, for dimension tables, is true when this column is the primary key of the actual table 502. The time navigation field, for the database dimension, is true if this field can be used by the time navigator.

Detailed Description Text (271):
The fact aggregate table 512 includes a list of fact aggregates in the datamart 150. The fact aggregates includes attributes that point to the actual fact table in

which this aggregate belongs. The fact aggregate table 512 indicates which numbered aggregate represents the fact table in question, the number of rows in this aggregate, a datamart letter, and an enabled flag. The datamart letter indicates the mirrored datamart to which this fact aggregate belongs.

Detailed Description Text (272):
Mirror is used to ensure that partially completed extractions from the source systems 110 do not cause the database to become inconsistent.

Detailed Description Text (274):
The dimension base aggregate table 516 lists all the dimension aggregates in the datamart. The dimension base aggregate includes the following attributes: an actual table key, an aggregate number, an aggregate size, a datamart letter, a dimension base aggregate key, and an enable flag. The actual table key points to the physical header for this dimension base. The aggregate number, for the dimension table in question, is the number of this particular aggregate. The aggregate size is the number of rows in the aggregate. The datamart letter indicates which mirrored database this aggregate lives in. The dimension base aggregate key is the primary key. The enable flag indicates whether or not the aggregate navigator should work with this aggregate.

Detailed Description Text (276):
The datamart letter 510 indicates which of two mirrored datamarts a particular aggregate belongs to. This is an optional element which may not be required if mirroring does not occur in the datamart 150. Mirroring duplicates the tables in the datamart 150. Changes can then be made to one copy of the datamart 150, while the other datamart 150 continues running. These changes can then be propagated when possible.

Detailed Description Text (277):
The actual column type 540 is a logical description of role a column plays in the system 100. The actual column type 540 includes attributes that define the default value to be used in a database for a column of this type.

Detailed Description Text (279):
The database physical type 595 defines the name of the physical database.

Detailed Description Text (280):
The translation actual table 539 defines the actual values of translations strings for a single relational database management system. These translations strings are the real strings to use for a given translation string within a store type. The translation actual table 539 also includes attributes that point to the store type.

Detailed Description Text (282):
The date table 560 is used to track date information in the datamart 150. Importantly, times and dates are always treated corrected in the datamart 150. This can be guaranteed because the consultant cannot change the definition of dates in the datamart 150. Thus, for example, the month of September will always have 30 days, and leap years will be handled correctly.

Detailed Description Text (285):
The business process table 570 is a look up table for supported business process types during the extraction. The business process 570 includes a business process key and a process name. The process name corresponds to a logical name for a business process to which fact staging table belongs. The process key identifies a business process record in a fact staging table.

Detailed Description Text (288):
The traditional solution in datamarts is to store periodic "snapshots" of the

balance. The snapshots are often stored at daily intervals for the recent historical past, and at greater intervals (e.g. weekly or monthly) for less recent history. This approach has two big disadvantages. The first is an enormous multiplication of data volume. If, for example, you are keeping track of inventory in a store you must store a snapshot for each product you hold in inventory for each day, even if you only sell a small fraction of all of your products on a given day. If you sell 10,000 different products but you only have 500 transactions a day, the "snapshot" datamart is 20 times larger than the transactional datamart. The second disadvantage relates to the most common solution for alleviating the first problem, namely storing snapshots at less frequent intervals for less recent history. This results in the inability to compare levels of inventory in corresponding time periods since the same level of detail is not present in earlier data. For example, in manufacturing companies it is often the case that much business is done near the end of fiscal quarters. If one wants to compare inventory levels between Q1 1995, Q1 1996 and Q1 1997, and focus on the most important changes which occur near quarter end, one cannot use the approach of storing the snapshots at coarser levels of detail since daily data would be required.

Detailed Description Text (289):
In some embodiments of the system, the aggregate tables are used to answer queries about backlog/balance/inventory quantities. In order to answer such queries, the previously described rolling forward from the beginning of time is done. However, this is performed efficiently through the accessing of the appropriate time aggregates. For example, assume the datamart 150 has five years of historical transaction data beginning in 1993. Assume that one desires the inventory of some specified products on May 10, 1996. This would be computed by querying all of the transactions in the 1993, 1994 and 1995 year aggregates, the 1996 Q1 quarter aggregate, the April 1996 month aggregate, the May 1996 week 1 aggregate and finally 3 days of actual May, 1996 daily transactions. These transactions (additions and subtractions from inventory) would be added to the known starting inventory in order to produce the inventory on May 10. Note that this "time navigation "hops" by successively smaller time intervals (year, quarter, month, week, day) in order to minimize the number of database accesses. What is important is the exploitation of aggregate tables, that already exist in the system in order to answer transactional queries rapidly (e.g. What were the total sales of product X in April 1996?). This avoids the need to build what is essentially a second data datamart with the balance/inventory/backlog snapshots.

Detailed Description Text (290):
Query Mechanism Metadata

Detailed Description Text (291):
The following describes the metadata 160 used in the query/reporting program 104. This metadata is shown in FIG. 6. Generally, the query mechanism metadata can be broken into ticksheet metadata, measurement metadata, filtering metadata and display options metadata. The ticksheet metadata defines the user interface objects for user interaction with the datamart 150. The ticksheet defines how users can initiate queries and how results are presented back to the user. The measurement metadata defines a logical business calculation that can be presented to a user. Typically, the measurement metadata defines a format for presenting information to user that is more easily understood by the user or provides a more valuable result to the user. The filtering metadata defines how a user can filter results. Filtering allows the results set to be limited to particular dimension values. The display options metadata defines display options that can be provided to the user.

Detailed Description Text (292):
The following describes some important features of the user interface. The user interface allows the user to drill down through data. Also, portions of the query forms can be dynamic based upon values in fields (e.g., a list box can be dynamically updated because it is tied to a field in the datamart 150, that when

changed, cause the values in the list box to change). Also, a query is guaranteed to be consistent with the schema because the query is tied to the schema definition.

Detailed Description Text (295):
Query Mechanism Schema Metadata Descriptions

Detailed Description Text (296):
Ticksheets Metadata

Detailed Description Text (299):
The ticksheet column 608 defines a single column for displaying measure choices on a ticksheet. The ticksheet column table 608 includes the ticksheet column key, the list order, the ticksheet key, and the description. These columns and the ticksheet column table 608 operate in a manner similar to such columns in other tables in this metadata.

Detailed Description Text (302):
The attribute table 610 defines the dimension attribute choices within a ticksheet. These choices are tied to a single dimension column in the schema definition of the datamart 150. The attribute table 610 includes an abbreviation, an attribute key, a dictionary key, a dimension column key, a dimension role key, a hyperlink, a label, a list order, a name, and a ticksheet key. The abbreviation is the shortened user string for the attribute. The attribute key is the primary key for this attribute. The dictionary key is a pointer to the dictionary 640 that includes help message for a particular attribute. The dimension column key is the dimension column in which this attribute refers. For degenerate dimensions this reference is null. The dimension usage, within a constellation, is defined by the dimension role key. The hyperlink is an html text for navigating return values for this attribute to other web sites, such as a company name look-ups etc. The label is what the user sees for a particular attribute. The list order defines a sort order on pop-up menus where one is the topmost in the list. The name is the internal name for the attribute. The ticksheet key indicates the ticksheet to which this attribute belongs.

Detailed Description Text (307):
Measurement Metadata

Detailed Description Text (308):
The following describes the measures used in the query mechanism schema 600. The measure table 620 defines a top level object for a logical business calculation. The measure table 620 includes a constellation key, a description, a measure key, a measure unit, and a name. The constellation key points to the constellation in which the measure resides. The description is for documentation purposes. The measure key is the primary key for the measure table 620. The measure unit is an indicator of the manner in which numbers are to be displayed. The name is the logical name of the measure.

Detailed Description Text (312):
Filtering Metadata

Detailed Description Text (314):
The following describes the filtering tables. The filter block 650 is a top level grouping table for filter area within a ticksheet. The filter block 650 is tied to a particular dimension column in the schema definition. The filter block 650 includes, columns, a description, a dictionary key, a dimension column key, a dimension role key, a filter block key, a filter block type, a label, a list order, a name, a plural, a mapping flag, and a ticksheet key. The columns field indicates the number of columns in this filter block. The description is for documentation. The dictionary key points to the help dictionary. The dimension column key points to the actual column name and the datamart to be filtered on. A null value here

means degenerate dimension as determined by the dimension role key. The dimension role key points to the dimension role in the constellation of the ticksheet that is the form key to filter on for all facts in this constellation. A null value here means that a special dimension shared by all constellations is being used. The filter block key is the primary key for this table. The filter block type points to the filter block type table 652 which defines the ways in which this filter block is displayed to the user (e.g., a checkbox or a radio button). The label is the text that appears to the user for the filter block. The list order is the order that the filter block should appear in a list. The name is the name of the filter block. The plural field is the text that appears to the user for the filter block. The mapping flag is used in mapping. The ticksheet key points to the ticksheet that this filter block belongs.

Detailed Description Text (316):
The filter element table 656 defines individual values for a dimension attribute within a filter block. The filter element table 656 includes a dictionary key, a filter element key, a filter group key, a label, a list order, a name, an SQL statement, and a value. The dictionary key points to the user help text for a particular filter element. The filter element key is the primary key. The filter group key points to the filter group to which this element belongs. The label is the user displayed string for the element. A list order is the order of this element within a filter group. The name is the hidden name of this element. The SQL statement is an SQL statement used to build the list of values for a dynamic list box filter group. The value is the database value that this element translates into in a SQL "WHERE" clause.

Detailed Description Text (317):
Display Options Metadata

Detailed Description Text (323):
User Interface Example of Defining Metadata

Detailed Description Text (325):
The following describes a constellation used in a business. In this example a new dimension is added very simply and the changes are automatically propagated into the datamart 150. The enterprise manager interface 192 is used by the consultant to define and manipulate the system 100.

Detailed Description Text (326):
FIG. 7 illustrates the enterprise manager interface 192. Multiple system 100's can be connected to through that interface. Many of the objects and tables in the system 100 are shown. The base dimensions definitions 710 correspond to the base dimensions available under the "epitest" datamart. The constellations 712 for this datamart include an expense constellation and a sales constellation 720. Thus, the sales constellation 720 would appear as a row in the constellation table 302. Under the sales constellation 720 appear the definitions for the sales aggregates 721, the sales dimensions 723, the sales degenerate dimensions 725, the sales facts 726, the sales measures 728, and the sales ticksheets 729. Also, the extraction definitions 740 and security definitions for the "epitest" datamart are accessible. The sales dimensions 723 define rows in the dimension role table 320. These rows include customer billed to, product, application, program, customer ship to, and territory.

Detailed Description Text (335):
FIG. 16 illustrates the results of a request by the consultant to generate the datamart 150 from the definitions of the datamart. The results show that a number of tables have been created in the datamart 150. Importantly, FIG. 16 illustrates the results of an initial build process. In subsequent modifications, only those elements of the datamart that have changed will be changed. In other words, the subsequent changes are handled as an update process. An example of the update

process is described below.

Detailed Description Text (337):
The following describes the creation of the connectors 162. Once the schema definitions 161 re set, the consultant then defines the connectors 162 to the source systems 110. The connectors, as noted above, define how information is to be extracted from the source systems 110 and how that information is to be placed into the datamart 150.

Detailed Description Text (340):
FIG. 19 illustrates the All Semantics connector as defined in the connector definition window 1900. This connector includes the description and a definition of the input and output data stores. In this case, both of the data stores are the "epimart" (which is the datamart 150).

Detailed Description Text (342):
Returning to the discussion of connectors, FIG. 21 illustrates the connector entitled MFG. The MFG connector has two major steps: (1) order dimension staging, and (2) order fact staging. The results of these extraction steps are put in the staging tables 130. (The all extraction steps window 2100 illustrates all the possible steps in the system that can be used.)

Detailed Description Text (343):
FIG. 22 illustrates the SQL statement window 2200. The SQL statement window 2200 has an SQL field 2210 that includes the SQL statements that loads a customer table. As shown in the dialog box, the table references for the SQL statement includes the customer dimension table column definitions. That is, this SQL statement is going to be used to populate the customer dimension table.

Detailed Description Text (344):
In this example, the base name, type code, type name, region code, region name and tier name corresponds to the column names within the customer dimension. The date modify is an additional field that is to be used to indicate when this field was last modified in the database. Additionally, there is a source system key that is automatically included in every dimension. The source system key helps ensure that the datainart 150 is well-formed.

Detailed Description Text (346):
FIG. 23 illustrates the SQL statement for the Open Order Stage for populating the order fact table.

Detailed Description Text (347):
At this point the steps for generating the staging table information are complete. Now the semantic conversion steps are defined.

Detailed Description Text (348):
In FIG. 24, returning to the connector steps window, we have switched to an All Semantics connector 2410. The All Semantics connector 2410 causes the semantic conversion of the information in the staging table for use in the datamart 150.

Detailed Description Text (351):
FIG. 27 illustrates the results of a consultant adding a new dimension 2700 (called warehouse) to the sales constellation 720. The batch operation window 1600 illustrates the changes that are being made to the datamart that was created in FIG. 16. To achieve these results, the consultant need only perform the following steps:

Detailed Description Text (353):
2. Define the connector steps, including the SQL Statement to extract the warehouse data from the source systems 110.

Detailed Description Text (356):
5. Have the enterprise manager 102 update the datamart 150.

Detailed Description Text (357):
Thus, changing the schema definition of the datamart 150 is significantly simpler than previous systems.

Detailed Description Text (366):
FIG. 34 illustrates another query form 3200 generated from a different ticksheet definition. When the user selects the create report button, the query is issued against the datamart 150. FIG. 35 illustrates some sample results from such a query.

Detailed Description Text (367):
The following query log illustrates the actual query that was executed against the datamart 150. The query log illustrates that an aggregate and navigation process determined which aggregate would be the most appropriate. The aggregate builder had created these aggregates. The most appropriate aggregate for the requested query was selected. The results were then returned.

Detailed Description Text (371):
Importantly, various embodiments of the invention do not necessarily include all of the features described above. For example, some embodiments of the invention do not include the first phase of the extraction process (loading the staging tables) because the source system data is provided to the system 100 directly by other extraction programs. Another example is where the datamart 150 is created, but a separate query interface is used to query the datamart 150. The query interface could use only a different communications protocol (e.g., instead of HTTP), or could be a completely different front end.

Detailed Description Text (373):
Some embodiments of the invention build a database system, not necessarily a datamart. Additionally, these embodiments do not have to conform to a star schema definition in the metadata 160.

Detailed Description Text (374):
An object database system could be generated instead of a relational database system.

Detailed Description Text (375):
Some embodiments of the invention comprise only a computer readable media (e.g., a CD, a tape, a hard drive or other storage media) that has the programs that implement all, or a portion of, the system 100. Some embodiments of the invention include an electromagnetic waveform having the programs. Some embodiments of the invention include only the computer system running the datamart, other embodiments of the invention include only the computer system that creates, accesses, and queries the datamart, but does not include in the datamart itself.

Detailed Description Paragraph Table (1):

```
******************************************************************** Query log
******************************************************************** time : <A
Date Here> addr : 192.0.0.210 host : 192.0.0.210 user : agent : Mozilla/4.01 [en]
(WinNT; U) Datebase information: DRIVER={SQL SERVER}; SERVER=bigfoot;
DATABASE=macromedia Keys and values coming in from the browser: file = fileDesc =
queryaction = QUERY hidden.sub.-- queryaction = QUERY OK.sub.-- callback =
NOK.sub.-- callback = ticksheet = Orders hidden.sub.-- ticksheet = Orders Rows =
Customer hidden.sub.-- Rows = Customer Columns = Fiscal Year hidden.sub.-- Columns
= Fiscal Year units = Price hidden.sub.-- units = Price facttype = Shipped
```

hidden.sub.-- facttype = Shipped facttype2 = Gross hidden.sub.-- facttype2 = Gross stage = Orders hidden.sub.-- stage = Orders currencyunits = Thousands hidden.sub.-- currencyunits = Thousands rowtotal = yes hidden.sub.-- rowtotal = yes columntotal = yes hidden.sub.-- columntotal = yes percent = none hidden.sub.-- percent = none precision = 0 hidden.sub.-- precision = 0 charts = 3D hidden.sub.-- charts = 3D maxrows = 10 hidden.sub.-- maxrows = 10 rowsorttype = value hidden.sub.-- rowsorttype = value Fiscal.sub.-- Years = All hidden.sub.-- Fiscal.sub.-- Years = All Fiscal.sub.-- Quarters = All hidden.sub.-- Fiscal.sub.-- Quarters = All Calendar.sub.-- Months = All hidden.sub.-- Calendar.sub.-- Months = All Business.sub.-- Units = All hidden.sub.-- Business.sub.-- Units = All Product.sub.-- Lines = All hidden.sub.-- Product.sub.-- Lines = All Product.sub.-- Supergroups = All hidden.sub.-- Product.sub.-- Supergroups = All Platforms = All hidden.sub.-- Platforms = All Product.sub.-- Languages = All hidden.sub.-- Product.sub.-- Languages = All Product.sub.-- SKUs = All hidden.sub.-- Product.sub.-- SKUs = All Product.sub.-- SKU = Sales.sub.-- Reps = All hidden.sub.-- Sales.sub.-- Reps = All Channels = All hidden.sub.-- Channels = All Customer.sub.-- Types = All hidden.sub.-- Customer.sub.-- Types = All Customer.sub.-- Regions = All hidden.sub.-- Customer.sub.-- Regions = All Customers = All hidden.sub.-- Customers = All Customer = sqStyle = classic hidden.sub.-- sqstyle = classic Contents of % FormData: Columns = Fiscal Year rowsorttype = value ticksheet = Orders Customers = All charts = 3D Customer Types = All Product SKUs = All Customer Regions = All Fiscal Quarters = All Rows = Customer Channels = All precision = 0 Product Supergroups = All percent = none maxrows = 10 Sales Reps = All columntotal = yes Calendar Months = All queryaction = QUERY sqStyle = classic Fiscal Years = All Product Languages = All Platforms = All Product Lines = All rowtotal = yes currencyunits = Thousands Business Units = All The colheaders are: The Cellitems are: Price Shipped Gross Orders The Cellitems abbreviated are: Dollar Amount Shipped Gross Orders pid is: 310 spid is: 25 The valid collheaders are: The invalid colheaders are: The valid cellitems are: Price Shipped Gross Orders The invalid cellitems are: The unitstack is: CURRENCY The cellstack is: SUN (Order.net.sub.-- price) The selectstack is: SUM (Order.net.sub.-- price) The typestack is SHIP Transtypes "BEGIN.sub.-- RETURN" = 1007 "END.sub.-- GROSS" = 1004 "END.sub.-- SRBOTH" = 1018 "END.sub.-- SRADJ" = 1012 "BOOK" = 1 "BEGIN.sub.-- ANET" = 1013 "END.sub.-- IADJ" = 1024 "END.sub.-- ICDNP" = 1022 "BEGIN.sub.-- GROSS" = 1003 "BEGIN.sub.-- SRBDTH" = 1017 "END.sub.-- SBOTH" = 1016 "END" = 1002 "BEGIN.sub.-- SRADJ" = 1011 "END.sub.-- SADJ" = 1010 "BEGIN.sub.-- IADJ" = 1023 "BEGIN.sub.-- ICOMP" = 1021 "END.sub.-- NET" = 1006 "SHIP.sub.-- ADJUST" = 103 "LOST" = 3 "END.sub.-- SALL" = 1020 "BEGIN.sub.-- SBOTH" = 1015 "BEGIN" = 1001 "BEGIN.sub.-- SADJ" = 1009 "BOOK.sub.-- RETURN" = 2 "END.sub.-- FADJ" = 1026 "BEGIN.sub.-- NET" = 1005 "BEGIN.sub.-- SALL" = 1019 "GL" = 105 "SHIP.sub.-- RETURN" = 102 "SHIP.sub.-- RADJ" = 104 "SHIP" = 101 "END.sub.-- RETURN" = 1008 "INV.sub.-- ADJUST" = 201 "LEAD.sub.-- LOST" = 4 "BEGIN.sub.-- FADJ" = 1025 "FINV.sub.-- ADJUST" = 202 "END.sub.-- ANET" = 1014 The facttable is: Order The limitvalues are: The access limitations are: The limitvalues are: The header took 261 milliseconds. Parsing took 160 milliseconds. Generating the header took 0 milliseconds. Building user limits took 0 milliseconds. Phase 0: Aggregate navigator initialization. Phase 1: Getting dimensions from SQL. (10) Phase 2: Getting fields available from SQL. (90) Phase 3: Getting degenerate fields from SQL. (0) Phase 0: Preparing for query building and execution.R2[0 .fwdarw. 0] = (SUM(Z0)) R1[0] = SUM(Z0) The column router: Cell location 0 will be returned in column 0 when Type is SHIP. The result router: Result location 0 is (SUM(Z0)) 0 The unique facttables are:Order The number of unique facttables are: 1 The unique types are:SHIP Table to unique number lookup Order => .sub.-- 0.sub.-- Begin work on the query based on the facttable Order Phase 1: Table Order creating table aliases (10) JOIN.sub.-- FIELD IS CUSTOMER.sub.-- BILLTO.sub.-- KEY FOR Customer JOIN.sub.-- FIELD IS FOR Fiscal Year SQL table aliases: Order.quadrature.: T3 Date.quadrature.: T2 Customer.quadrature.CUSTOMER.sub.-- BILLTO.sub.-- KEY : T1 Table aliases: tablealiaslookup(T1) = Customer tablealiaslookup(T2) = Date tablealiaslookup(T3) = Order selectalias: Customer: T1.base.sub.-- name Fiscal Year: T2.fy.sub.-- name selectstackalias: SUM(Order.net.sub.-- price): -SUM(T3.net.sub.-- price) joinalias:

Customer: T1.customer.sub.-- key = T3.customer.sub.-- billto.sub.-- key Phase 2: Building SELECT clause (0) Phase 3: Building FROM clause (0) Phase 4: Building WHERE clause (0) Phase 5: Building GROUP BY clause (0) SQL before going through the aggregate navigator: SELECT Columns = T2.fy.sub.-- name, Type = T3.Transtype.sub.`3 key, SUM(T3.net.sub.-- price), Rows = T1.base name INTO #tmp.sub.-- 0.sub.-- FROM Customer T1, Date T2, Order T3 WHERE T1.customer.sub.-- key = T3.customer billto.sub.-- key and T2.date.sub.-- key = T3.date.sub.-- key and T3.Transtype.sub.-- key in (101) GROUP BY T1.base.sub.-- name, T2.fy.sub.-- name, T3.Transtype.sub.-- key
*** Selecting appropriate aggregate for the query.
*** Phase 0 Aggregate navigator. Preparing for query building and execution. Phase 1 Spliting query into clauses. (0) Phase 2 Construction of aliases. (0) Phase 3 Extracting neededfields from where clause. (0) Phase 4 Extracting neededfields from group by clause. (0) Phase 5 Extracting neededfields from select clause. (0) Phase 6 Unaliasing. (0) Phase 7 Constructing the SQL to fetch smallest aggregate. (20) Phase 8 Running the big SQL. (20) Phase 9 Extracting results from the big SQL. (0) Phase 10 Adjusting input with aggregate information. (0) Phase 6: Aggregate Navigating (40)
*** Appropriate aggregate determined (CUSTOMER.sub.-- 0; DATE.sub.-- 4,

Other Reference Publication (2):
Tsuda, K. et al., "IconicBrowser: An Iconic Retrieval System for Object-Oriented Databases", Proc. of the IEEE Workshop on Visual Languages, Oct. 4, 1989, pp. 130-137.

Other Reference Publication (4):
Kimball, R., "The Data Warehouse Toolkit", (1996) John-Wiley & Sons, Inc., 388 pages (includes CD ROM).

Other Reference Publication (9):
Kawaguchi, A. et al., "Concurrency Control Theory for Deferred Materialized Views", Database Theory-ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 306-320.

Other Reference Publication (10):
Zhuge, Y. et al., "Consistency Algorithms for Multi-Source Warehouse View Maintenance", Distributed and Parallel Databases, vol. 6, pp. 7-40 (1998), Kluwer Academic Publishers.

Other Reference Publication (13):
Yang, J. et al., "Maintaining Temporal Views Over Non-Historical Information Sources For Data Warehousing", Advances in Database Technology--EDBT '98, Proceedings of the 6th International Conference on Extending Database Technology, Valencia, Spain, Mar. 1998, pp. 389-403.

Other Reference Publication (14):
Quass, D., "Maintenance Expressions for Views with Aggregation", Proceedings of the 21st International Conference on Very Large Data Bases, IEEE, Zurich, Switzerland, (Sep. 1995), 9 pages.

Other Reference Publication (15):
Mumick, I. et al., "Maintenance of Data Cubes and Summary Tables in a Warehouse", Proceedings of the 1997 ACM SIGMOD International Conference, ACM Press, 1997, pp. 100-111.

Other Reference Publication (16):
Huyn, N., "Multiple-View Self-Maintenance in Data Warehousing Environments",

Proceedings of the 23rd International Conference on Very Large Data Bases, IEEE, (1997), pp. 26-35.

Other Reference Publication (18):
Gupta, H. "Selection of Views to Materialize in a Data Warehouse", Database Theory--ICDT '97, Proceedings of the 6th International Conference, Delphi, Greece, Jan. 1997, pp. 98-112.

Other Reference Publication (21):
Labio, W. et al., "Physical Database Design for Data Warehouses", IEEE Paper No. 1063-6382/97, IEEE (1997), pp. 277-288.

CLAIMS:

1. A method of generating a datamart having aggregates using a computer system, the method comprising:

accessing a schema definition which describes a schema for the datamart;

accessing a description of a set of aggregates to be generated in the datamart;

generating a set of commands from the schema definition, including,

generating a set of table creation commands, and

generating a set of table access and manipulation commands, the set of table access and manipulation commands corresponding to the semantic meaning of the schema;

generating a set of generate aggregates commands; and

generating a set of aggregate tables using the set of generate aggregates commands.

2. The method of claim 1 further comprising accessing data in the datamart, the description of the schema, and the description of the set of aggregates to populate the set of aggregate tables.

7. A method of querying a datamart comprising:

accessing a definition of a schema for the datamart and a definition of a set of aggregates for the datamart;

generating the set of aggregates for the datamart from the definition of the schema and the definition of the aggregates;

determining at least an aggregate of the set of aggregates that most closely corresponds to the query, the query corresponding to the schema definition;

querying the datamart for the data corresponding to the aggregate.

8. The method of claim 7 wherein the definition of the set of aggregates is stored in a set of tables in a database, and wherein determining the aggregate includes performing a view on the set of tables to determine a possible set of aggregates that can be used to answer the query.

11. A system for generating a datamart having aggregates, the system comprising:

a data store for storing a description of a schema for the datamart;

a first program for accessing a description of a set of aggregates to be generated in the datamart, the first program further for generating a set of commands from the schema definition, the first program further for generating a set of table creation commands, and for generating a set of table access and manipulation commands, the set of table access and manipulation commands corresponding to the semantic meaning of the schema;

a second program for generating a set of generate aggregates commands, and the second program for generating a set of aggregate tables using the set of generate aggregates commands.

Previous Doc Next Doc Go to Doc#